

## Introduction

Building and maintaining the integrity of FISCAL's products and it's customers data is of the upmost importance. The Engineering team need to ensure that they design, build and maintain their solutions to ensure that security of our customer's information and data is a key focus.

This page covers various elements of application security that should be considered while designing, building and maintaining FISCAL's solutions. Members of the Engineering team should ensure that they remain familiar with it's contents.

## Application Development Security Assessments

Every Story is reviewed and sized in a High Level Planning (HLP) session prior to implementation. During this review a security assessment will be performed to identify any potential security risks. If a security issue is identified a solution will be agreed and documented in the Story to either address or mitigate the risk. The Story will also be marked with the "Security" tag to highlight an issue has been identified.

When assessing a feature or story for potential security risks as a minimum the following areas should be considered:

- How data can be stored securely ensuring that it is only accessed by those processes and individuals that require it
- How data is transported securely across networks and between processes ensuring access is restricted to those that need it
- Will the potential solution be vulnerable to any of the current OWASP web application security risks (see below)
- How integrating additional third party libraries or packages could potentially impact the solution's security and what can be done to mitigate any potential risks
- How environments can be created in a secure and repeatable way to ensure that they are brought up identically each time (Infrastructure as code) with a view to minimise manual access and monitoring

In the event a potential security concern is discovered during the implementation of the story an assessment should take place at that point with at least one of the team's senior engineers present. Again the results of the assessment should be recorded in the story and the "Security" tag should be added.

When reviewing the completed story, the reviewer should ensure that recommendations identified during the security risk assessment have been implemented.

## OWASP Web Application Security Risks

While developing software it is important to be aware of potential security risks, so that as to not introduce potential vulnerabilities into our software solutions. To assist in developing secure software the OWASP Foundation publishes a list of the top 10 web application security risks.

The OWASP Foundation exists to make software security visible, so that individuals and organisations are able to make informed decisions. The OWASP top 10 is a list of the most commonly found and exploited web application vulnerabilities. The list is compiled from feedback from leading vendors and professionals working in the industry. The list is intended to raise awareness amongst developers and managers. It is updated every three years and was last published in 2021.

Currently the Foundation identifies the following security risks in its top 10:

1. [A01:2021 - Broken Access Control](#)
2. [A02:2021 - Cryptographic Failures](#)
3. [A03:2021 - Injection](#)
4. [A04:2021 - Insecure Design](#)
5. [A05:2021 - Security Misconfiguration](#)
6. [A06:2021 - Vulnerable and Outdated Components](#)
7. [A07:2021 - Identification and Authentication Failures](#)
8. [A08:2021 - Software and Data Integrity Failures](#)
9. [A09:2021 - Security Logging and Monitoring Failures](#)
10. [A10:2021 - Server-Side Request Forgery](#)

The up to date list along with a detailed description of each issue and how to prevent them can found [here](#). Engineers should monitor the list to maintain familiarity with current threats and keep aware of modifications.

## Access to Customer Data & Sites

By default members of the software development team do not have access to live customer sites or data, but occasionally team members may require access to investigate and diagnose issues.

Access to a customer site can be obtained by making a request to the Customer Support Team. They are able to grant access to the site on a temporary basis. Software Engineers must inform the Customer Support team as soon as they have completed their investigation and access is no longer required.

Members of the Software Development and Data teams can self-manage access to NXG Forensics customer data via the Customer Management tool, by selecting the required customer and locating the Data Access panel. The Engineer should enter the reason they require access (including a ZenDesk or Azure DevOps reference) and then select "Grant access". This will grant the requester temporary read access to the customers data which will automatically be revoked after 24 hours. Each request is logged to maintain an audit record of every request.

If access is required to an AP Forensics customer then a request should be made to the IT Team via IT Helpdesk. Once the investigation is complete and access is no longer required the IT Helpdesk ticket should be updated so that the access can be revoked.

## Monitoring Application Security

Regular security vulnerability scans will be run across Fiscal's active software and the third party package dependencies it is built on. When the scan results identify potential vulnerabilities they will be assessed to determine how quickly they need to be addressed. This assessment will be based on the severity of the issue and whether the it can practically impact the security of the software.

In terms of severity these fall into the following four categories:

- Critical
- High
- Medium
- Low

Generally critical and high severity issues will be addressed at the earliest opportunity. This will ideally be in the current sprint, which may include releasing a hot-fix if required, as long as the fix can be released safely and without jeopardising the stability of the software. If this is not the case then a resolution should be available in the following sprint.

Medium and low severity issues will be scheduled into the short-term road-map to ensure that they are addressed at the earliest opportunity. Typically a resolution to these issues should be delivered in the next 1 to 2 sprints.

Note, in the majority of cases the vulnerability is likely to exist in a third party package. In which case the resolution will be addressed once an updated version of the package is available.

## **Local Administration Rights for Engineer's Workstations**

Members of the Software Development team are granted local Admin rights to their individual machines so that they can manage the software installed on them.

This allows them to install and maintain the development tools and packages required to develop FISCAL's software solutions. This privilege allows the engineers to perform their roles effectively causing minimum disruption to the internal IT team.

This approach has been accepted by the SIRI/CISO, as it is perceived that given the development processes and precautions in place (as identified above) the risk of a member of the Engineering team causing a security breach either deliberately or inadvertently is minimal and this far outweighs the inconvenience and disruption to productivity caused by requiring a member of the IT Team to install it on their behalf.

Members of the Engineering team are required to take all sensible precautions to ensure they maintain the security and integrity of their individual workstations. Any potential issues or security vulnerabilities should be reported immediately to the IT Team and any misuse or abuse of this privilege will result in it being revoked; either for the individual or the team collectively.

- [Summary](#)
- [RAG](#)
- [Azure Cognitive Search](#)
- [Azure Cognitive Search OpenAI Demo](#)
  - [Limitations of the Demo and Fiscal Data](#)
- [Azure OpenAI Studio](#)
- [Findings](#)
- [Next Steps](#)
- [Notes](#)
- [Resources](#)

## Summary

ChatGPT was augmented with Fiscal data, such as the Customer Security Document, so that it could answer questions such as "*Does NXG encrypt data?*"

The purpose of this page is to present the findings of experiments with Azure Cognitive Search to create a Retrieval-Augmented Generation (RAG) Chatbot.

## RAG

RAG chatbots combine the best of retrieval-based and generative models. While retrieval-based models search through a database to find the most relevant answer, generative models can create answers dynamically.

RAG is a pattern that augments the capabilities of a Large Language Model (LLM) like ChatGPT by adding an information retrieval system that provides the data. Adding an information retrieval system gives you control over the data used by an LLM when it formulates a response. For an enterprise solution, RAG means that you can constrain natural language processing to your enterprise data sourced from vectorized documents, images, audio, and video.

A high-level summary of the pattern looks like this:

- Start with a user question or request (prompt).
- Send it to Azure Cognitive Search to find relevant information.
- Send the top ranked search results to the LLM.
- Use the natural language understanding and reasoning capabilities of the LLM to generate a response to the initial prompt.
- Azure Cognitive Search provides inputs to the LLM prompt, but doesn't train the model. In RAG architecture, there's no extra training. The LLM is pretrained using public data, but it generates responses that are augmented by information from the retriever.

## Azure Cognitive Search

Microsoft has multiple options for using Azure Cognitive Search to implement the RAG pattern, including:

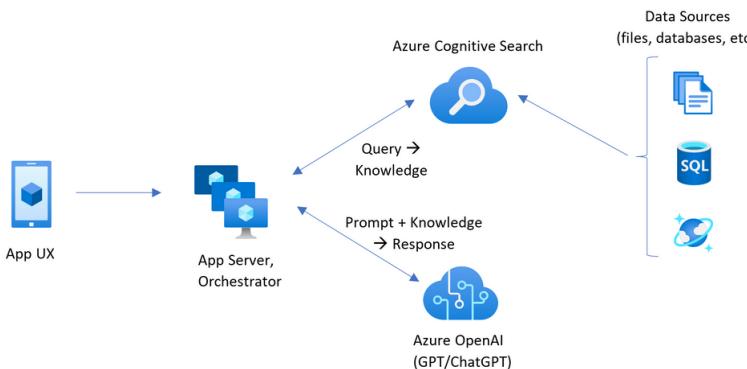
- [Azure Cognitive Search OpenAI Demo](#), a custom RAG solution.
- [Azure OpenAI Studio](#), using your data with an [Azure OpenAI Service](#). The studio integrates with Azure Cognitive Search for storage and retrieval. If you already have a search index, you can connect to it in the studio and start chatting right away. If you don't have an index, you can create one by uploading your data using the studio.
- [Azure Machine Learning](#), a search index can be used as a vector store. You can create a vector index in an Azure Machine Learning prompt flow that uses your Cognitive Search service for storage and retrieval.

## Azure Cognitive Search OpenAI Demo

Most of the experiments (and all of the findings) were performed using the Azure Cognitive Search OpenAI Demo. It uses the Azure OpenAI Service to access the ChatGPT model (defaults to gpt-35-turbo).

The demo uses these elements:

- App UX (web app) for the user experience.
- App server or orchestrator (integration and coordination layer).
- Azure Cognitive Search (information retrieval system).
- Azure OpenAI (LLM for generative AI).



The demo was run virtually using GitHub Codespaces, which opens a web-based VS Code in your browser.

Although the demo includes sample data, it was modified to use Fiscal data such as the Customer Security Document, Statement Reconciliation docs and a completed customer security questionnaire.

If you would like to repeat the experiments then please note that the most time-consuming process is likely to be obtaining an **Azure subscription with access enabled for the Azure OpenAI service** - this requires you to request access from Microsoft. Approval can take up to 10 business days but in this case it only took 1 day.

## Limitations of the Demo and Fiscal Data

The demo has a few limitations:

- Data ingestion is by PDF only.
- There is no persistent chat history.

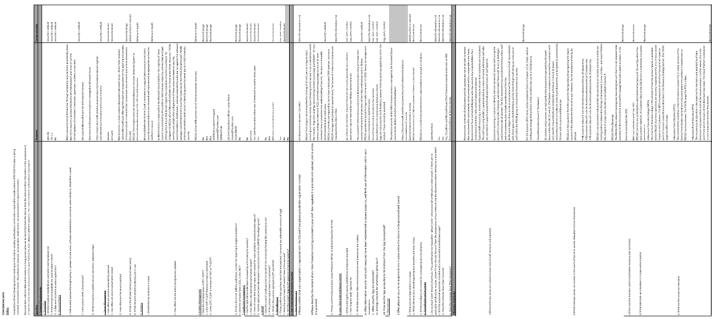
The Fiscal data has a few limitations:

- The data is generally unstructured and [Azure Form Recogniser](#) (used by Azure Cognitive Search) often could not pair questions and answers.

### Data Protection and Confidentiality

1. What information will the organisation be processing, handling or storing on behalf of CNTW? Does this include special category information (e.g. physical and mental health, race/ethnic origin etc.)?	The purpose of this project is to implement FISCAL NXG Forensics Software Service. The software aims to improve the financial operational performance of the organisation by: <ol style="list-style-type: none"><li>1. automatically and continuously analysing payments data (typically invoices and credit notes) prior to payment to determine the risk of error and/or fraud and for reporting compliance to payment processes;</li><li>2. (optionally) more efficiently supporting audit recovery projects of previously enacted payments; and</li><li>3. (optionally) automatically and continuously analysing the Master Supplier File for opportunities to reduce the risk of error and/or fraud caused by duplicated supplier records, supplier records that are not actively being used or changes.</li></ol>
2. Who is the data controller for the information?	CNTW
3. If applicable, provide an approximation of the number of records processed or held over the lifetime of the contract.	This is fully dependent on the quantity of data NTW choose to present to the product
4. Where will the information be held? Is the	Electronic format. The NXG Forensics service is hosted on

- Conversion from Word/Excel to PDF made it even more difficult for Azure Form Recogniser.



- Not enough data!

## Azure OpenAI Studio

A RAG Chatbot was also created using the studio - this was straightforward and produced similar results (with similar costs) to the Azure Cognitive Search OpenAI Demo.

## Findings

Where the Fiscal data could be extracted by Azure Form Recogniser, the responses were good.

What does statement reconciliation do?

Statement reconciliation is a module provided by FISCAL Technologies that allows users to upload supplier statements directly to a supplier page in NXG. The software automatically matches the statements and alerts users to any issues or discrepancies for review. Users can quickly resolve errors and recover cash from overpayments or unclaimed credits [\[1\]](#) [\[2\]](#).

Citations: [1. Statement Reconciliation: FISCAL Technologies](#) [2. Statement Reconciliation Fact Sheet-0.pdf](#)

Does Fiscal have a DPO?

According to the information provided, FISCAL is currently not required by law to have a dedicated Data Protection Officer (DPO) [\[1\]](#).

Citations: [1. FISCAL - ISMS - Frequently Asked Questions-11.pdf](#)

Does NXG encrypt data?

Yes, NXG Forensics encrypts data. All data transfers and data at rest are encrypted using AES-256 encryption. Data in transit is transmitted over HTTPS using TLS 1.2 encryption. [\[1\]](#)

Citations: [1. FISCAL - ISMS - Frequently Asked Questions-15.pdf](#)



The development approach used by Fiscal Technologies for its application NXG Forensics is based on Agile software development techniques, including Scrum, Test Driven Development, Pair Programming, Continuous Integration, and Continuous Delivery. The development process involves separate test and production environments hosted on Microsoft Azure, tracking of identified changes with assigned work items, modifications made in two-week sprints and separate branches, promotion of verified changes to a release at the end of each sprint, and review and approval of the release by the CAB (Change Advisory Board) <sup>1</sup>.

Citations: [1. Customer Security Document-5.pdf](#)

who are the fiscal development team?



The fiscal development team is composed of permanent employees located in the United Kingdom. They are responsible for the development and verification of the NXG Forensics Solution <sup>2</sup>. The development team follows an Agile approach and uses techniques such as Scrum, Pair Programming, Continuous Integration, and Continuous Delivery <sup>2</sup>. They develop the solution using the Microsoft .NET platform and the source code is held in a source repository <sup>2</sup>. Access to live production environments is granted to members of the development team on a limited basis, only when required to investigate and diagnose issues <sup>3</sup>.

Citations: [1. Customer Security Document-6.pdf](#) [2. FISCAL - ISMS - Frequently Asked Questions-30.pdf](#)

[3. Customer Security Document-13.pdf](#)

## However, where the Fiscal data was present but could not be extracted, the responses were poor.

what is fiscal's DISASTER RECOVERY plan?



The information provided does not specifically mention the details of Fiscal's disaster recovery plan. Therefore, I don't have access to the specific information about their disaster recovery plan.

What firewalls have been installed?



I don't have information about which firewalls have been installed. <sup>1</sup> <sup>2</sup> <sup>3</sup>

Citations: [1. csirt-2.pdf](#) [2. FISCAL - ISMS - Frequently Asked Questions-38.pdf](#)  
[3. CNTW External parties info security questionnaire Ref 2022-62-7.pdf](#)

## A disappointing lack of knowledge of acronyms.

What are Fiscals SLOs?



There is no information available about "Fiscals SLOs" in the provided sources.

SLOs?



I'm sorry, but I couldn't find any information about SLOs in the provided sources.

Service Level Objectives?



The Service Level Objectives for remedial software support are as follows <sup>1</sup>:

- Severity 1: Initial Response Time - 1 business hour; Solution Target Time - 4 business hours
- Severity 2: Initial Response Time - 2 business hours; Solution Target Time - 1 business day
- Severity 3: Initial Response Time - 4 business hours; Solution Target Time - 3 business days
- Severity 4: Initial Response Time - 1 business day; Solution Target Time - Reasonable endeavors

Please note that these response and resolution times are estimates and actual times may vary <sup>1</sup>.

Citations: [1. FISCAL Support Services Specification-0.pdf](#)

what does the acronym SLO mean?



There is no information available in the provided sources about the acronym SLO.

Are all personnel required to sign non-disclosure or confidentiality agreements

Based on the provided information, it is unclear whether all personnel are required to sign non-disclosure or confidentiality agreements. [\[1\]](#) states that non-disclosure or confidentiality agreements are not required for all personnel, but they can be signed at the request of the customer. However, [\[1\]](#) also mentions that FISCAL's contract contains confidentiality requirements, which may indicate that employees of FISCAL are required to sign confidentiality agreements. It is recommended to consult the specific company's employment contract or HR policies to determine the exact requirements regarding non-disclosure or confidentiality agreements.

Citations: [1. CNTW External parties info security questionnaire Ref 2022-62-2.pdf](#)

The citations stuff is pretty cool (you just click on a citation link in a response).

Thought process Supporting content Citation

Searched for: NXG credit monitoring

Conversations:

```
{"role": "system", "content": "Assistant helps the company employees with their healthcare plan questions, and questions about the employee handbook. Be brief in your answers.\nAnswer ONLY with the facts listed in the list of sources below. If there isn't enough information below, say you don't know. Do not generate answers that don't use the sources below. If asking a clarifying question to the user would help, ask the question.\nFor tabular information return it as an html table. Do not return markdown format. If the question is not in English, answer in the language used in the question.\nEach source has a name followed by colon and the actual information, always include the source name for each fact you use in the response. Use square brackets to reference the source, e.g. [info1.txt]. Don't combine sources, list each source separately, e.g. [info1.txt][info2.pdf].\n\n"}  
{"role": "user", "content": "Does NXG do credit monitoring?"}, {"source": "NXG Software Specification September 2023-1.pdf", "text": "Pending actions can be viewed and shared internally or sent to the supplier. A full audit log tracks changes and updates. Credit Monitoring. FISCAL is Credit Monitoring module allows continuous monitoring of all active suppliers' credit scores information across your supply chain down to the individual supplier level. NXG continuously checks all active suppliers (up to your agreed volume cap) for the most up to date credit scores and will alert any suppliers whose scores are below the customer defined threshold. Alerts can immediately be investigated by users using the current and historic credit scores, supplier spend detail provided and any other risk information held in the given supplier dashboard. The software is hosted in the cloud and presented as a SaaS service. Use of the service is subject to the prevailing FISCAL terms and conditions. NXG Forensics provides an API for appropriate data to be injected into the service. The provision of the data to be processed via the API is subject to the client's acceptance of the terms and conditions of the service. The client must confirm that all CNTW information contains personal information will be handled in accordance with Data Protection legislation? Page 1 of 6 The purpose of this project is to implement FISCAL NXG Forensics Software Service. The software aims to improve the financial operational performance of the organisation by: 1. automatically and continuously analysing payments data (typically invoices and credit notes) prior to payment to determine the risk of error and/or fraud and for reporting compliance to payments processes; 2. (optionally) more efficiently supporting audited payments; 3. (optionally) automatically and continuously analysing supplier records that are not actively being used or changes. CNTW This is fully dependent on the quantity of data that is chosen to present to the product Electronic format. The NXG Forensics service is hosted on secure, highly available and resilient Tier 3 (or better) Microsoft Azure datacentres.\nIMS - Frequently Asked Questions -7.pdf: payments data (typically invoices and credit notes) prior to payment to determine the risk of error and/or fraud and for reporting compliance to payments processes; more efficiently supporting audit recovery projects previously enacted payments; and continuously analysing supplier records that are not actively being used or changes. The risk of error and/or fraud caused by duplicated supplier records, supplier records that are not actively being used or changes, NXG Forensics utilises a ready-only extract from the organisation's ERP/accounting system. This will be collected, typically daily, and uploaded (automatically or manually) into the FISCAL NXG Forensics software service. The extracted data will be encrypted both in transit and when stored within the NXG Forensics platform and it will remain in the same geo-regulatory region as specified during implementation. It will not be shared in an unencrypted form with anyone other than FISCAL. The Artificial Intelligence enabled processing compares each payment and/or supplier record with stored historic and current records identifying potentially abnormal ('exception') characteristics with the payment Confidential !"}, {"role": "user", "content": "What is the cost of running the NXG credit monitoring service?"}, {"source": "NXG Software Specification September 2023-1.pdf", "text": "Azure Applied AI Services: £2.76 Cognitive Services: £0.02 Storage: <£0.01"}, {"source": "Azure Cognitive Search", "text": "Azure Cognitive Search: £0.55 App Service: <£0.01 Bandwidth: <£0.01"}  
The cost of running the NXG credit monitoring service depends on the usage of various Azure services. The main cost is for Azure Cognitive Search, which is £0.55 per unit. There is also a cost for Azure App Service and Bandwidth, both of which are less than £0.01 per unit.
```

## Next Steps

- Restructure the Fiscal data to be extractable by Azure Form Recogniser.
- Use more Fiscal data, such as a curated employment contract.
- Re-run the demo.
- Check for knowledge of synonyms.
- Brainstorm other uses for RAG Chabots, e.g. helpdesk, product summaries, product help, supplier portal.
- Look at alternatives such as [Llamaindex](#).
- Look at mechanisms to add new data and update existing data.
- How to apply permissions to data.
- Review security and performance before deploying to production.

## Notes

- Xelix are probably using RAG for their AI helpdesk module. All they would need is a suitably formatted helpdesk doc.
- The Fiscal data is not available in public ChatGPT, it is only available in Fiscal's augmented ChatGPT, so no leakage of Fiscal IP.

Azure Applied AI Services	Azure Cognitive Search
£2.76	£0.55
Cognitive Services	App Service
Storage	Bandwidth
<£0.01	<£0.01

- Costs to deploy and run Azure Cognitive Search OpenAI Demo for 1/2 day with about 20 queries:
- Access to Azure OpenAI is currently limited. Customers that wish to use Azure OpenAI must submit a registration form for both initial experimentation access, and again for approval for use in production.

## Resources

[Understanding RAG a simple guide.pdf](#)

## ⌚ An example that has annoying code

In the following example we have remembered to execute `AsUtc` on our first query, but unfortunately we've forgotten to do it on the second one. This means when we show that date in the UI, it'll assume it's in the user's local time zone, so it could be quite wrong depending on where the user is!

! Reminder: Not all dates should be UTC, if it comes from the user and we don't know what time zone it *should* be, leave it alone (like dates in the user's uploads); but if it comes from us (like the date/time a user did something), it should be UTC!

```
using System;
using Util;

public sealed record MyObject(int Id, string Name, DateTime MyDate);

var query = (await db.QueryAsync<MyObject>(sql))
    .Select(x => x.MyDate.AsUtc()); // UTC

// loads of other code

var anotherQuery = await db.QueryAsync<MyObject>(otherSql); // not UTC :(
```

## ⌚ On the record

If we instead put the call to `AsUtc` in the record definition as follows, we only need to do it once. Every time we create an instance of our `MyObject` record the date will automatically have the UTC date kind specified.

When it's serialised via the API it will output something like `2023-07-05T16:51:12Z`, which is an ISO UTC date time, therefore when we use `Date` or `moment` in typescript, it'll interpret that date correctly.

The two attributes here are needed because we are overriding the `MyDate` property that is included in the record by default, so we need to tell tools that we actually want to include our copy:

- `[Required]` - This ensures any versions of this object serialised via Swagger will have this date included.
- `[JsonPropertyName]` - This ensures when the object is serialised to Hangfire it will have this date included.

We also make our overridden `MyDate` property `readonly` so it behaves the same as the original property, as the values on a `record` should be immutable and prevents Sonar from complaining about public properties!

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Text.Json.Serialization;
using Util;

public sealed record MyObject(int Id, string Name, DateTime MyDate)
{
    [Required]
    [JsonPropertyName("MyDate")]
    public readonly DateTime MyDate = MyDate.AsUtc();
}

var query = await db.QueryAsync<MyObject>(sql); // date will be UTC automatically
var anotherQuery = await db.QueryAsync<MyObject>(otherSql); // here too!
```

## ⌚ Classy

We can do exactly the same thing using a class instead of a record, but as we're not overriding a default property we don't need the attributes.

```
using System;
using Util;

public sealed class MyObject
{
    private DateTime _myDate;

    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime MyDate { get => _myDate; set => _myDate = value.AsUtc(); }

    var query = await db.QueryAsync<MyObject>(sql); // date will be UTC automatically
    var anotherQuery = await db.QueryAsync<MyObject>(otherSql); // here too!
```

## ⌚ Jobs won't die

If there is a long running step in your Hangfire job, you'll find if you come to cancel it in the UI it'll sit there dutifully executing until it reaches a point where it'll cancel. By default, these points will be when our code ceases to execute and control is handed back to Hangfire; so essentially not until the job completes (or fails). Not ideal when we have a job stuck that's not going to finish for a long time!

```
public sealed class MyHangfireJob
{
    public async Task ExecuteJob(IEnumerable<string> myThings)
    {
        await DoSomethingTimeConsuming();

        foreach(var thing in myThings)
        {
            await DoSomethingElseTimeConsuming(thing);
        }
    }
}
```

## ⌚ Cancellation

First of all we need to pass a cancellation token to our job and then check if cancellation is requested in our loop. That way if cancellation occurs before we complete iterating over the things in our loop, an exception is thrown and we terminate early.

```
public sealed class MyHangfireJob
{
    public async Task ExecuteJob(IEnumerable<string> myThings, CancellationToken cancellationToken)
    {
        await DoSomethingTimeConsuming();

        foreach(var thing in myThings)
        {
            cancellationToken.ThrowIfCancellationRequested();

            await DoSomethingElseTimeConsuming(thing);
        }
    }
}
```

## ⌚ Share the token

Even better, if we have long running operations that could also perform their own cancellation checking we can pass the token along. That way if cancellation occurs while we're still in one of those methods, we don't have to wait for them to complete and reach the next check in our for loop!

Many methods in library code already support this, for example have a look at `Task.Delay` as it allows cancellation if you just pass the token along; Visual Studio will even tell you when you're not passing it to a method that could take it, very useful!

```
public sealed class MyHangfireJob
{
    public async Task ExecuteJob(IEnumerable<string> myThings, CancellationToken cancellationToken)
    {
        await DoSomethingTimeConsuming(cancellationToken);

        foreach(var thing in myThings)
        {
            cancellationToken.ThrowIfCancellationRequested();

            await DoSomethingElseTimeConsuming(thing, cancellationToken);
        }
    }
}
```

## ⌚ Scheduling the job

When it comes to scheduling the job, you don't actually need to have access to the cancellation token, already you can just pass `CancellationToken.None` and Hangfire will do the rest:

```
var jobId = HangfireClient.Enqueue<IMyJob>(
    job => job.DoThing(myParam, CancellationToken.None)
);
```

## ⌚ Testing when cancellation occurs

It's pretty straightforward to write a test to check the cancellation token is working as well!

```
public sealed class MyHangfireJobTests
{
    private readonly MyHangfireJob _subject = new();

    [Fact]
    public async Task ExecuteJob_Cancels()
    {
        using var cancellationTokenSource = new CancellationTokenSource();

        var action = () => _subject.ExecuteJob(
            new[] { "towel", "babel fish", "Hitchhiker's Guide to the Galaxy" },
            cancellationTokenSource.Token
        );

        cancellationTokenSource.Cancel();

        await action.Should().ThrowAsync<OperationCanceledException>();
    }
}
```



## Structure of a UI test

There are a few things that are worth noting about how to structure a UI test:

- **Assertions:** Asserting the element has the value you expect uses the aptly named `expect` method. `expect` has a whole host of helper methods to check things are in the document, they have specific HTML properties like disabled and more! i.e. `expect(element).toBeInTheDocument()`.
- `import "@testing-library/jest-dom";`: Just include this import in your tests, it makes checking for the *absence* of elements (i.e. `expect(element).not.toBeInTheDocument()`) possible.
- **Finding elements:** Take a look at [the docs](#) for more details!
  - `queryByX`: There are several methods, like `queryByRole`, `queryByText`, `queryByPlaceholder` etc. These will look for an element in the generated HTML, if it's there it'll return it if not it'll return null. We can use this to check if something is or isn't in the document.
  - `findByX`: Performs the same function as `queryByX` but it's async and it'll error if it's not found.
  - `getByX`: Performs the same function as `queryByX` but it's not async and it'll error if it's not found.
  - **Why can't I query/find by CSS selector, id etc.:** The testing library uses non-implementation details for the selectors; this is important to ensure your tests will not break with irrelevant changes; this is why you can't search by id, or a CSS selector. If you find yourself wanting/need to do that, you probably should change the component so you don't!
  - `data-testid`: Why not just put `<button data-testid="submit" value="Submit" />` on everything? Well now you're adding testing specific code to the product, which will be included in the bundle that we use for production; this should be unnecessary, so avoid!
- **What if my element takes a moment to stabilise?:** In many cases you may find your element doesn't have the state you expect immediately, this is especially true after performing an action. We can use the `waitFor` method which will keep checking the generated HTML until it becomes the state we expect *or* the timeout occurs which will fail the test.

```
await waitFor(() => {
  const element = screen.queryByRole("button", { name: /Submit/ });
  expect(element).toBeInTheDocument();
});
```

- **Actions:** One of the obvious things to test is the user interactivity in the component. There are two main things you might want to test, clicking and typing:
  - **Clicking:** We need to find the component, check it's there and click it:

```
const element = screen.queryByRole("button", { name: /Submit/ });
expect(element).toBeInTheDocument();

fireEvent.click(element!);

// now check whatever affect you expect clicking the button to have on the HTML

◦ Typing: We need to find the component, check it's there and type in it:
const element = screen.queryByPlaceholder("Search");
expect(element).toBeInTheDocument();

await act(() => userEvent.type(element!, "My text"));

// now check whatever affect you expect typing the text to have on the HTML
```

## ⌚ Writing UI tests is annoying...

Let's say I want to test this component, the obvious thing to test is typing into our search box which should search the user list for an id:

```
import { useCustomerShortName, useNxgClient, useUnitName } from "@fiscaltec/api.sdk";
import { useQuery } from "@tanstack/react-query";
import { ChangeEvent, useCallback, useState } from "react";

const MyUserList = () => {
  const customer = useCustomerShortName();
  const unit = useUnitName();

  const api = useNxgClient();

  const [searchText, setSearchText] = useState("");

  const onChange = useCallback((event: ChangeEvent<HTMLInputElement>) => setSearchText(event.target.value), []);

  const { data } = useQuery([customer, unit, "users"], () => api.unitUsers_GetUsers(customer, unit), { enabled: searchText !== "" });

  return (
    <>
      <input type="text" value={searchText} onChange={onChange} placeholder="Search" />

      {data && (
        <ul>
          {data?.users
            ?.filter(user => user.id === searchText)
            .map(user => (
              <li key={user.id}>{user.name}</li>
            )))
        </ul>
      )}
    </>
  );
};
```

I have to wrap it with these other components so the test doesn't complain about all the things it needs to be mocked; in this case that includes:

- The URL path (`MemoryRouter`) as it needs to extract the unit "fiscal" from it.
- The `QueryClient` (`QueryClientProvider`) for `react-query`.
- The NXG client which is the API (`NxgClientContext.Provider`). This is actually the only thing we're providing any customisation to at all.
- The customer context (`CustomerContext.Provider`) i.e. the customer short name we usually extract from the URL.
- The route (`Routes -> Route`) so it knows when to render our test component, and to say where the unit placeholder is which will be populated with the unit from `MemoryRouter`.

```
import { CustomerContext, INXGClient, NxgClientContext } from "@fiscaltec/api.sdk";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import "@testing-library/jest-dom";
import { act, render, screen, waitFor } from "@testing-library/react";
import userEvent from "@testing-library/user-event";
```

```

import { MemoryRouter, Route, Routes } from "react-router";

describe("MyUserList", () => {
  test("Searches for user", async () => {
    const queryClient = new QueryClient();

    const api = {
      unitUsers_GetUsers: async (_, __, ___) => ({
        users: [
          {
            id: "1st",
            name: "tester1@fiscaltec.com",
          },
          {
            id: "2nd",
            name: "tester2@fiscaltec.com",
          },
        ],
      }),
    } as INXGClient;

    render(
      <MemoryRouter initialEntries={["/units/fiscal/users"]}>
        <QueryClientProvider client={queryClient}>
          <NxgClientContext.Provider value={api}>
            <CustomerContext.Provider value="fiscal">
              <Routes>
                <Route path="/units/:unit/users" element={<MyUserList />} />
              </Routes>
            </CustomerContext.Provider>
          </NxgClientContext.Provider>
        </QueryClientProvider>
      </MemoryRouter>
    );
  });

  const list = screen.queryByRole("list");
  expect(list).not.toBeInTheDocument();

  const search = screen.queryByPlaceholderText("Search");
  expect(search).toBeInTheDocument();

  await act(() => userEvent.type(search!, "2nd"));

  await waitFor(() => {
    const list = screen.queryByRole("list");
    expect(list).toBeInTheDocument();

    let row = screen.queryByText("tester2@fiscaltec.com");
    expect(row).toBeInTheDocument();

    row = screen.queryByText("tester1@fiscaltec.com");
    expect(row).not.toBeInTheDocument();
  });
});
});

```

## ⌚ How should I setup my tests?

Setting up UI tests can be annoying, the issue is we need to mock/provide everything that our React component is expecting to be there including all the context our hooks need; unfortunately this can be a lot of stuff as we have contexts, URL parameters, contexts from libraries like `react-query`... the list goes on!

Instead of doing all this setup for **every** single component we want to test, we can override the `render` method to provide a wrapper, this wrapper can then include all this setup with some sensible default values so we only have to override the parts that actually matter. We have a component that does all of this work for you, the `render` method exported from [testComponents](#).

Here's that same test using our custom `render` method, in this case the only customisation we need to pass in is our `api` mock; and the rest of the test is the same!

```

import { INXGClient } from "@fiscaltec/api.sdk";
import "@testing-library/jest-dom";
import userEvent from "@testing-library/user-event";
import { act, render, screen, waitFor } from "./testComponents";

describe("MyUserList", () => {
  test("Searches for user", async () => {
    const api = {
      unitUsers_GetUsers: async (_, __, ___) => ({
        users: [
          {
            id: "1st",
            name: "tester1@fiscaltec.com",
          },
          {
            id: "2nd",
            name: "tester2@fiscaltec.com",
          },
        ],
      }),
    } as INXGClient;

    render(<MyUserList />, { api });

    const list = screen.queryByRole("list");
    expect(list).not.toBeInTheDocument();

    const search = screen.queryByPlaceholderText("Search");
    expect(search).toBeInTheDocument();

    await act(() => userEvent.type(search!, "2nd"));

    await waitFor(() => {
      const list = screen.queryByRole("list");
      expect(list).toBeInTheDocument();
    });
  });
});

```

```
expect(list).toBeInTheDocument();

let row = screen.queryByText("tester2@fiscaltec.com");
expect(row).toBeInTheDocument();

row = screen.queryByText("tester1@fiscaltec.com");
expect(row).not.toBeInTheDocument();
});

});
```

## Structure of a hook test

- **renderHook:** Use the `renderHook` method to render our hook and give us access to anything it returns using its `result.current` property.
  - **Assertions:** Assert that any of those properties are the value we expect. This is important for both default values, and after some action that should change them.
  - **act:** When we perform some action, like calling a callback that will change the results wrap the call in `act`:
- ```
const { result } = renderHook(() => useHook());
act(() => result.current.myCallback("some data"));

What if the hook takes a moment to stabilise?: Similar to the component example we can use waitFor to wait for the assertion to be true, or fail if it does not become true within the timeout.
```
- Why not just make a component use the hook?**: Obviously you can absolutely do this, but then you don't have access to the things the hook returns.
1. If we're testing a hook that could be/is used elsewhere we ought to unit test it and that's much easier if you can access its properties.
  2. Using a component you'd need to write the values out to the DOM and this gets very messy, very quickly, trust me I did it...
  3. There's a wrapper for `renderHook` in [testComponents](#) which can setup routing, `react-query` etc. just like we do for a component test, but without the component!

## Test your hooks!

Let's say I wanted to test the following hook which restricts some input text to a specific length, returning how many characters we have remaining to type and the percentage of our allowable characters we have used:

```
import { ChangeEvent, useCallback, useMemo, useState } from "react";

export default function useDoHookStuff(maxLength = 100) {
  const [text, setText] = useState("");
  const onChange = useCallback((event: ChangeEvent<HTMLInputElement>) => setText(event.target.value.substring(0, maxLength)), [maxLength]);
  const { remainingSize, percentage } = useMemo(
    () => ({
      remainingSize: maxLength - text.length,
      percentage: text.length / maxLength,
    }),
    [maxLength, text.length]
  );
  return {
    text,
    onChange,
    remainingSize,
    percentage,
  };
}
```

We could test this by testing the component where this is used, but let's say for arguments sake that this hook is used by several components and we'd like to unit test it independently. We ought to test all the functionality of this hook, that is the current text being updated, the new remaining and percentage being calculated and the text being forcibly limited to the maximum. As follows is a test that covers all 3 of these features as well as that sensible default values are used initially:

```
import { act, renderHook } from "@testing-library/react-hooks";
import { ChangeEvent } from "react";

describe("useDoHookStuff", () => {
  test("defaults", () => {
    const { result } = renderHook(() => useDoHookStuff());
    expect(result.current.text).toBe("");
    expect(result.current.remainingSize).toBe(100);
    expect(result.current.percentage).toBe(0);
  });

  test("partially full", () => {
    const { result } = renderHook(() => useDoHookStuff(5));
    act(() => result.current.onChange({ target: { value: "te" } } as ChangeEvent<HTMLInputElement>));
    expect(result.current.text).toBe("te");
    expect(result.current.remainingSize).toBe(3);
    expect(result.current.percentage).toBe(0.4);
  });

  test("overflow", () => {
    const { result } = renderHook(() => useDoHookStuff(5));
    act(() => result.current.onChange({ target: { value: "overflow" } } as ChangeEvent<HTMLInputElement>));
    expect(result.current.text).toBe("overf");
    expect(result.current.remainingSize).toBe(0);
    expect(result.current.percentage).toBe(1);
  });
});
```

## What about those sneaky dependencies?

As alluded to earlier, you can use the same wrapper mechanism to setup routing, `react-query`, `current user` etc. the same as we do when testing a component using our custom `RenderHook` method in [testComponents](#).

The following hook and its test shows how we can use it to succinctly write the setup that'll allow us to use a mock to verify our API endpoint is called with the correct data.

Hook:

```
import { useCustomerShortName, useNxgClient, useUnitName } from "@fiscaltec/api.sdk";
import { useMutation } from "@tanstack/react-query";
```

```
export default function useMakeRequest() {
  const customer = useCustomerShortName();
  const unit = useUnitName();

  const api = useNxgClient();

  return useMutation(({ supplierRef, erpId, comment }: { supplierRef: string; erpId: number; comment: string }) =>
    api.statements_PostTransactionComment(supplierRef, erpId, { comment }, customer, unit)
  );
}
```

Test:

```
import { INXGClient } from "@fiscaltec/api.sdk";
import { act, renderHook } from "./testComponents";

describe("useMakeRequest", () => {
  test("saves", async () => {
    const mockCall = jest.fn();

    const api = {
      statements_PostTransactionComment: mockCall,
    } as unknown as INXGClient;

    const { result } = renderHook(() => useMakeRequest(), { api, customer: "customer", unit: "unit" });

    await act(async () => {
      await result.current.mutateAsync({ supplierRef: "123", erpId: 4, comment: "something" });
    });

    expect(mockCall).toHaveBeenCalledWith("123", 4, { comment: "something" }, "customer", "unit");
  });
});
```

When adding new `react-query` queries have a think about how the key should be used, generally speaking we need the following:

1. **Hierarchical:** Reading from left to right we should go from the largest thing, to the smallest. That way if we want to invalidate many things at once we can invalidate on a higher key. i.e. `["suppliers", erpId, supplierRef, "statements", "documents", filter]` we can invalidate at `["suppliers", erpId, supplierRef]` and all the queries for that supplier will invalidate including the statement ones! If we invalidate at `["suppliers", erpId, supplierRef, "statements", "documents"]` it'll invalidate all the statement documents, regardless of the `filter`.
2. **Include Parameters** - The query key **has** to include all parameters that are variable, if it doesn't the query cache may think this query is the same as something else and return the data from a different query!
3. **Written in one place** - If you're copying/pasting the keys, or worse re-writing them, it's easy to make a mistake or someone to miss your copy when they need to change the key in the future. I recommend you read [this](#) about why "magic strings" are bad, some of these don't entirely apply to typescript but they do for this example. This applies to C# as well, and I think it's something we fall foul of more often than we ought to.
4. Read [react-query Query Key Guide](#) and [Effective React Query Keys](#)!

## ⌚ Magic, Magic Strings

This example breaks all the rules we mentioned above! 😊 It also has a bug because our query key is wrong in the invalidation, this won't error but it also won't do anything useful!

```
export function useQueryMarbles(erpId: number, supplierRef: string, param1: string, param2: string) {
  const api = useNwgClient();

  return useQuery(["suppliers", "marbles", erpId, supplierRef], () => api.suppliers_GetMarbles(erpId, supplierRef, param1, param2));
}

export function useMutateMarbles(erpId: number, supplierRef: string) {
  const api = useNwgClient();
  const queryClient = useQueryClient();

  return useMutation((data: MyData) => api.suppliers_SetMarbles(erpId, supplierRef, data), {
    onSuccess: () => queryClient.invalidateQueries(["suppliers-Marbles"]),
  });
}
```

## 🔧 Factory to the rescue

The following example ensures we have no magic strings as all the strings are defined once in their appropriate functions, plus the hierarchy is enforced by each more specific key referencing its less specific parent key!

**Note:** We perform a filter in the `marbles` key generator function, this allows us to invalidate on the `marbles` key without the values of `param1` and `param2` being defined.

```
export class SuppliersKeyFactory {
  static base = ["suppliers"];

  static supplier = (erpId?: number, supplierRef?: string) => [...this.base, erpId, supplierRef];

  static marbles = (erpId?: number, supplierRef?: string, param1?: string, param2?: string) =>
    [...this.supplier(erpId, supplierRef), "marbles", param1, param2].filter(key => key !== undefined);
}

export function useQueryMarbles(erpId: number, supplierRef: string, param1: string, param2: string) {
  const api = useNwgClient();

  // ["suppliers", erpId, supplierRef, "marbles", param1, param2]
  return useQuery(SuppliersKeyFactory.marbles(erpId, supplierRef, param1, param2), () => api.suppliers_GetMarbles(erpId, supplierRef, param1, param2));
}

export function useMutateMarbles(erpId: number, supplierRef: string) {
  const api = useNwgClient();
  const queryClient = useQueryClient();

  return useMutation((data: MyData) => api.suppliers_SetMarbles(erpId, supplierRef, data), {
    onSuccess: () => queryClient.invalidateQueries(SuppliersKeyFactory.marbles(erpId, supplierRef)),
  });
}
```

## ⌚ An example that has a potential performance issue

In the following example we have a component that contains an input text box with a reset and submit button. The change, reset and submit actions each have their own functions. However, like the [useMemo example](#), these functions will be redefined every time this component re-renders. This can happen because any of the props to the component change, or because one of the hooks results change, and some other reasons too. Check with the React debugger in the browser, it may surprise you how often this happens!

```
import { ChangeEvent, useState } from "react";

const MyComponent = () => {
  const submit = useMutation();

  const initialValue = "Howdy";
  const [value, setValue] = useState(initialValue);

  const onChange = (event: ChangeEvent<HTMLInputElement>) => {
    event.stopPropagation();
    setValue(event.target.value);
  };

  function doSomeStuff(value: string) {
    let newValue = doSomeOtherStuff(value);
    newValue = doEvenMoreStuff(newValue);
    return newValue;
  }

  const onSubmit = async () => {
    const submitValue = doSomeStuff(value);
    await submit.mutateAsync(submitValue);
  };

  return (
    <>
      <input type="text" value={value} onChange={onChange} />
      <button value="Reset" onClick={() => setValue(initialValue)} />
      <button value="Submit" onClick={onSubmit} />
    </>
  );
};
```

## ⌚ Using useCallback

In this case these functions are simple, so the performance argument here is somewhat contrived, but there are many functions so they will soon add up! There is no harm in using `useCallback` so let's switch it round so those sneaky functions aren't redefined!

If you're not entirely familiar with `useCallback` you can read about it on [W3Schools](#) and [React's own documentation](#).

As you can see, wrapping our functions in `useCallback` is simple, and doesn't really make the code much more difficult to read. The functions will now only re-define when their dependencies change, for all but `onSubmit` they have no dependencies so they're not going to be re-defined at all!

Also note how `onReset` doesn't have a dependency on `initialValue`, that's because `initialValue` is a constant. We know it's not going to change between renders, so when I wrote this in VS Code the linter knew not to add it to the dependency list, nice!

```
import { ChangeEvent, useCallback, useState } from "react";

const MyComponent = () => {
  const submit = useMutation();

  const initialValue = "Howdy";
  const [value, setValue] = useState(initialValue);

  const onChange = useCallback((event: ChangeEvent<HTMLInputElement>) => {
    event.stopPropagation();
    setValue(event.target.value);
  }, []);

  const onReset = useCallback(() => setValue(initialValue), []);

  const doSomeStuff = useCallback((value: string) => {
    let newValue = doSomeOtherStuff(value);
    newValue = doEvenMoreStuff(newValue);
    return newValue;
  }, []);

  const onSubmit = useCallback(async () => {
    const submitValue = doSomeStuff(value);
    await submit.mutateAsync(submitValue);
  }, [doSomeStuff, submit, value]);

  return (
    <>
      <input type="text" value={value} onChange={onChange} />
      <button value="Reset" onClick={onReset} />
      <button value="Submit" onClick={onSubmit} />
    </>
  );
};
```

## ⌚ Alternatives?

If your function doesn't depend on any properties inside your component then you don't need to `useCallback` at all, you can simply define your function outside your component and it simply won't re-define on each render.

```
import { ChangeEvent, useCallback, useState } from "react";

const MyComponent = () => {
```

```
const submit = useMutation();

const initialValue = "Howdy";
const [value, setValue] = useState(initialValue);

const onChange = useCallback((event: ChangeEvent<HTMLInputElement>) => {
  event.stopPropagation();
  setValue(event.target.value);
}, []);

const onReset = useCallback(() => setValue(initialValue), []);

const onSubmit = useCallback(async () => {
  const submitValue = doSomeStuff(value);
  await submit.mutateAsync(submitValue);
}, [submit, value]);

return (
  <>
    <input type="text" value={value} onChange={onChange} />
    <button value="Reset" onClick={onReset} />
    <button value="Submit" onClick={onSubmit} />
  </>
);
};

function doSomeStuff(value: string) {
  let newValue = doSomeOtherStuff(value);
  newValue = doEvenMoreStuff(newValue);
  return newValue;
}
```

## ⌚ When shouldn't I use it

The documentation provides some good examples about how you can make most uses of `useCallback` unnecessary, but there's also no real harm in using it. Have a read through this as it gives some other great performance pointers!

[React docs deep dive "Should you use `useCallback` everywhere?](#)

## ⌚ An example that has a potential performance issue

In the following example we pull some data from the API, and then perform some expensive operation on that data. However, like the [useCallback example](#), that expensive operation per row will be re-executed every single time this React component re-renders. This can happen because any of the props to the component change, or because one of the hooks results change, and some other reasons too. Check with the React debugger in the browser, it may surprise you how often this happens!

```
const MyComponent = () => {
  const { data } = useQuery();

  const results = data.map(row => expensiveOperation(row));

  return (
    <>
      {results.map(row => (
        <>{row.value}</>
      ))}
    </>
  );
};
```

## ⌚ Using useMemo

So what can we do to fix this? Why `useMemo` of course!

If you're not entirely familiar with `useMemo` you can read about it on [W3Schools](#) and [React's own documentation](#).

Essentially what we need to know for this example, is that if we wrap the expensive operation with `useMemo` it will memoize (or cache) that value, only recomputing it if our dependencies (that's the list of values in the second parameter `[data]`) changes. In this case, it'll only recompute when `data` changes, which makes sense as when the result of the API changes we'll need to recompute the results!

```
import { useMemo } from "react";

const MyComponent = () => {
  const { data } = useQuery();

  const results = useMemo(() => data.map(row => expensiveOperation(row)), [data]);

  return (
    <>
      {results.map(row => (
        <>{row.value}</>
      ))}
    </>
  );
};
```

## ⌚ What about pulling out a bunch of stuff?

In the scenario where you need to pull out several properties from the same object (`data` in this case), you can return a dictionary from `useMemo`.

I wouldn't do this if each of the properties had separate dependencies, as recalculating would cause both those expensive operations to be repeated, but as it's just `data` in this case which will change atomically this is ideal!

```
import { useMemo } from "react";

const MyComponent = () => {
  const { data } = useQuery();

  const { value1, value2, value3 } = useMemo(
    () => ({
      value1: data.thing1,
      value2: expensiveOperation(data.thing2),
      value3: anotherExpensiveOperation(data.thing3),
    }),
    [data]
  );

  return (
    <>
      {results.map(row => (
        <>{row.value}</>
      ))}
    </>
  );
};
```

## ⌚ When shouldn't I use it

The documentation provides some good examples about how you can make most uses of `useMemo` unnecessary, but there's also no real harm in using it. Have a read through this as it gives some other great performance pointers!

[React docs deep dive "Should you use useMemo everywhere?"](#)

## When should I use a reducer?

Take a look at [React's documentation](#).

A reducer works similarly to `setState`, however it allows you to nominate a dispatch function which will be used when updating the state returned by `useReducer`. This is useful in the following scenarios (but not limited to these):

- If you have a complex state object, which multiple forms/inputs can update.
- You want to keep the manipulation of a state object in a single place so it's easy to see what manipulations it can undergo.
- There are many actions you could take to update state, like adding/removing/resetting etc.!

Although it can be used anywhere `useState` is used, in most cases it's more work to utilise it, so think about what the benefits are!

## Using useState

As follows is an example which highlights the reasons given above that `useReducer` can improve our code. Although, I'm sure you can see other ways to improve it, this is based on an example I saw recently which I have simplified here for the purposes of demonstration; if you're interested take a look at [ListEditor](#).

There's a few limitations with our design which fixing will necessitate increasing the complexity 😞:

- Each one of our callbacks has to perform 2 state manipulations (and the reset has to make 3), removing from `things` and updating `likes/dislikes`.
- We can't move from `likes` to `things`, or `dislikes` to `things`, or `likes` to `dislikes` or even `dislikes` to `likes`!
- If we wanted to make our list a re-usable component, we'd have to pass all 3 list states and all 3 set states.
- If we wanted to add more actions, like sorting or other moving rules we'd need to have even more callbacks.

```
import { useCallback, useState } from "react";

const initial = ["cats", "dogs", "penguins", "spiders", "dinosaurs"];

const MyLists = () => {
  const [things, setThings] = useState(initial);
  const [likes, setLikes] = useState<string[]>([]);
  const [dislikes, setDislikes] = useState<string[]>([]);

  const moveToLikes = useCallback((thing: string) => {
    setLikes(current => [...current, thing]);

    setThings(current => {
      const newThings = [...current];
      const index = newThings.indexOf(thing);
      if (index !== -1) {
        newThings.splice(index, 1);
      }
      return newThings;
    });
  }, []);

  const moveToDislikes = useCallback((thing: string) => {
    setDislikes(current => [...current, thing]);

    setThings(current => {
      const newThings = [...current];
      const index = newThings.indexOf(thing);
      if (index !== -1) {
        newThings.splice(index, 1);
      }
      return newThings;
    });
  }, []);

  const reset = useCallback(() => {
    setThings(initial);
    setLikes([]);
    setDislikes([]);
  }, []);

  return (
    <>
      <ul>
        <caption>things</caption>
        {things.map(thing => (
          <li key={thing} onClick={() => moveToDislikes(thing)} onDoubleClick={() => moveToLikes(thing)}>
            {thing}
          </li>
        ))}
      </ul>

      <ul>
        <caption>likes</caption>
        {likes.map(thing => (
          <li key={thing}>{thing}</li>
        ))}
      </ul>

      <ul>
        <caption>dislikes</caption>
        {dislikes.map(thing => (
          <li key={thing}>{thing}</li>
        ))}
      </ul>
    </>
    <button onClick={reset}>Reset</button>
  );
}
```

```
};  
export default MyLists;
```

## Using useReducer

Now let's switch the example to use a reducer!

As you can hopefully see we've made some fundamental changes:

- Merged the state into a single object, we could've done this for the previous example, but it would've made the state manipulation even more difficult to follow!
- Used a single `move` function which utilises the reducer's `MoveAction` to move from a `target` from a `source` list to a `destination` list. We've got the ability to move from/to any list with no extra code!
- Updated the `reset` function to use the reducer's `ResetAction`.
- Instead of using `const [state, setState] = useState(initialState);` we setup a reducer `const [state, dispatch] = useReducer(dispatchFunction, initialState);`

The reducer itself takes just 2 arguments

- The first argument, `current` is the current state, so at first it will be all our options in the `things` list with nothing in `likes` or `dislikes` as this is the initial state as passed to the second argument of `useReducer`.
- The second argument, `action` is the action we'd like to perform, specifically the `MoveAction` or `ResetAction`. This is simply the object we passed to `dispatch` in our callbacks.
- It returns the new state object after whatever manipulations the action should take on it.
- **! Note:** If you're manipulating objects or lists, remember to keep them immutable!
  - For an object, instead of just `current.property = newValue` do `const newState = { ...current, property: newValue };`!
  - For a list, instead of just `current.splice(index, 1, newValue)`; do `const newState = [...current]; newState.splice(index, 1, newValue);`!

```
import { useCallback, useReducer } from "react";  
  
const initial = ["cats", "dogs", "penguins", "spiders", "dinosaurs"];  
  
type ListState = {  
    likes: string[];  
    dislikes: string[];  
    things: string[];  
};  
  
type ListName = keyof ListState;  
  
const MyLists = () => {  
    const [state, dispatch] = useReducer(reducer, { things: initial, likes: [], dislikes: [] });  
  
    const move = useCallback((key: ListName, thing: string, destination: ListName) => dispatch({ type: "Move", source: key, destination, target: thing }), [state]);  
  
    const reset = useCallback(() => dispatch({ type: "Reset" }), [state]);  
  
    return (  
        <>  
            {Object.keys(state)  
                .map(key => key as ListName)  
                .map(key => (  
                    <ul key={key}>  
                        <caption>{key}</caption>  
                        {state[key].map(thing => (  
                            <li key={thing} onClick={() => move(key, thing, "dislikes")} onDoubleClick={() => move(key, thing, "likes")}>  
                                {thing}  
                            </li>  
                        ))}  
                    </ul>  
                ))}  
            <button onClick={reset}>Reset</button>  
        </>  
    );  
};  
export default MyLists;  
  
type MoveAction = { type: "Move"; source: ListName; destination: ListName; target: string };  
type ResetAction = { type: "Reset" };  
  
type ListAction = MoveAction | ResetAction;  
  
function reducer(current: ListState, action: ListAction) {  
    if (action.type === "Reset") {  
        return {  
            things: initial,  
            likes: [],  
            dislikes: [],  
        };  
    } else if (action.type === "Move") {  
        const source = [...current[action.source]];  
        const destination = [...current[action.destination]];  
  
        const index = source.indexOf(action.target);  
        if (index !== -1 && action.source !== action.destination) {  
            source.splice(index, 1);  
            destination.push(action.target);  
  
            return {  
                ...current,  
                [action.source]: source,  
                [action.destination]: destination,  
            };  
        }  
    } else {  
        throwUnknownAction(action); // compile-time error if any action types are not handled  
    }  
  
    return current;  
}  
  
function throwUnknownAction(_: never) {
```

```
    throw new Error("Unknown action type");
}
```

This section provides definitions for each of the development stages required to complete a sprint story or task.

## "To Do"

A story/task can be moved into the current sprint's "To Do" column the once following criteria are met:

- The stories acceptance criteria been reviewed and agreed with the Product Owner
- The story has been broken into tasks if required
- The task acceptance criteria has been checked by an Architect
- The tasks have been sized and are each less than 8 story points
- The story is tagged with [Comms] if a "What's new" entry is required

## "Doing"

A story/task can be moved into the "Doing" column the once following criteria are met:

- The story/task is currently in the "To Do" column
- There is a free lane on the board (WIP limit is number of developers + 1)

Once moved to the "Doing" column the card should be reflect the following information:

- The ADO state should indicate that story/task is in "Doing"
- The story/task should be updated with the date it was moved into "Doing"
- The assignee should reflect the person working on the story/task

## "Reviewing"

A story/task can be moved into the "Reviewing" column the once following criteria are met:

- Once all work is complete to fully satisfy the acceptance criteria
- Relevant and meaningful automated tests have been created to test the changes
- The developer has tested the changes on their development environment
- All current automated tests run successfully
- A Pull Request has been submitted for the changes and related tests
- Appropriate documentation complete

Once moved to the "Reviewing" column the card should be reflect the following information:

- The ADO state should indicate that story/task is in "Reviewing"
- The story/task should be updated with the date it was moved into "Reviewing"

## "Worthyable"

A story/task can be moved into the "Worthyable" column the once following criteria are met:

- All related Pull Requests have been accepted
- The changes have been successfully built, deployed and verified on Hallway
- All current automated tests run successfully
- The developer has tested the changes on Hallway
- The "What's new" entry (if required) has been drafted
- The impact on performance has been assessed

Once moved to the "Worthyable" column the card should be reflect the following information:

- The ADO state should indicate that story/task is in "Worthyable"
- The story/task should be updated with the date it was moved into "Worthyable"

## "Done"

A story/task can be moved into the "Done" column the once following criteria are met:

- The Product Owner has reviewed it and confirms that it satisfies all the specified acceptance criteria
- Any deferred defects have been raised in ADO and accepted by the Product Owner
- The "What's new" entry (if required) is ready to be deployed
- The story/task is ready to be deployed to the live environment

Once moved to the "Done" column the card should be reflect the following information:

- The ADO state should indicate that story/task is in "Closed"
- The story/task should be updated with the date it was moved into "Done"

- [Summary](#)
- [Suggested Format](#)
- [Date and Venue](#)
- [Suggested Topics for Breakout Sessions](#)
- [Questions](#)

## Summary

The purpose of this page is to plan a day where the whole of Engineering meet to think about the bigger picture, and to inspect and adapt our **working practices**.

We will discuss some **product improvements** when we meet but that's not the main focus.

## Suggested Format

One or more of these [retro ideas](#) and this meeting structure:

- Intro
- Breakout sessions #1
- Presentations #1
- Lunch
- Breakout sessions #2
- Presentations #2
- Wrap up

## Date and Venue

- Thursday 30th November (the day before the all-hands / Christmas party) at [The Curious Lounge](#).

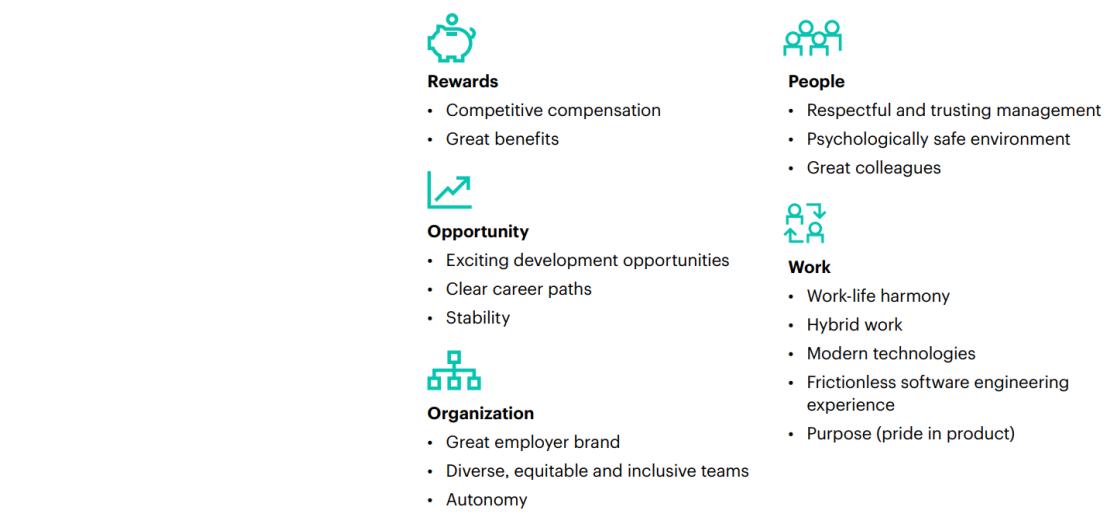
## Suggested Topics for Breakout Sessions

We won't be able to cover all of these topics, so we will need to decide which are the highest priority.

Example outcomes for the sessions: decisions, recommendations, summaries, ideas for further research.

To make the most of the sessions it will be beneficial to research the topics that interest you beforehand.

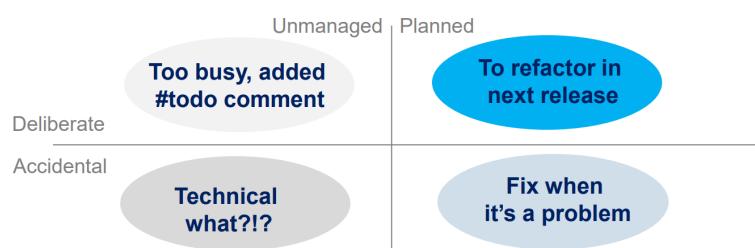
- Possible moonshots, e.g.
  - Massive reduction in false positives.
  - Massive reduction in time to onboard customers.
  - A convincing Galvanising Story.
  - Use of ML/AI through [RAG Chatbots](#) and [Prompt Engineering](#) (as used by Dan's custom report tiles).
  - Finding new business areas / creating a network effect.
- Creating "positive tension"
  - Between Product and Engineering, i.e. building the right product v building the product right.
  - Between Product and IT, mainly around [Separation of Duties](#).
- How to deal with Azure resource contention
  - Temporary scale up of resources.
  - Configure the max degree of parallelism (SQL's MAXDOP).
- Innovation opportunities
  - How do we carve out more time for innovation?
  - Possibly use Amazon's "Working Backwards" approach to assess feasibility.
- [ML/AI opportunities](#)
- Mass dismiss
- xunit
  - It's going to be used more soon and it's been neglected since it was released.
- Professional development
  - Technical skills.
  - Soft skills.
- What makes a compelling developer experience?



- Extracted from a recent Gartner report:
- [Software Engineering Role Progressions](#)

- Setting OKRs and measuring their progress
- Customer pain-points
- Supporting the wider business
- Avoiding stressful sprints
  - Avoiding API spikes - perhaps stress testing?
  - Integration or performance tests for Biblioteca.
  - Include history in our performance tests (revisions and workflow).

- Tech debt

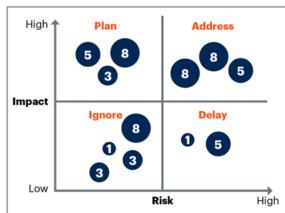


- [Gartner - How to Manage Tech Debt](#)

## How Do We Manage Tech Debt As A Business?

- Turn accidental tech debt into a deliberate, informed business decision

- Build a better software factory



Gartner

- Vitally integration

- Common identifier across NXG, Zendesk, Outlook, Salesforce and Xero.
- NXG data.
- [Vitally integrations](#), e.g. REST API v PostgreSQL for NXG data.

- Metrics

- [Software Development Metrics](#).
- System health metrics.

- Building a more reliable product

- [Non-Functional Requirements \(NFRs\)](#), such as stability, scalability and observability.

- Reducing developer cognitive load

- IDE GenAI plug-ins and Conversational AI, e.g. GitHub Copilot, Amazon CodeWhisper, Replit, Codeium.

- Is the generated code accurate, secure, performant, conformant and licensed? How do we prevent the leakage of Intellectual Property?
- How do we refine the skill of "prompt design", i.e. the combo of natural language and coding practices to optimise code generation with minimal effort?
- What is the monthly cost per developer?
- Gartner prediction: By 2027, 70% of professional developers will use AI-powered coding tools, up from less than 10% in 2023.
- [Exploit Generative AI for Happier, More Productive Software Developers](#) ← requires registration to view
- OpenAI partnered with GitHub to create GitHub Copilot, which they call an AI pair programmer. GitHub Copilot integrates the power of OpenAI Codex into a plugin for developer environments like Visual Studio Code.

- Context switching.
- Disaster Recovery
- Integrating Data Science findings into NXG
- Team health check

- How is [Squad Health Check Model](#) going?

- See example from 5th May (the notes are pictures from the previous time... which would have been pre-covid!)

| Squad Health Check |                                                                                             | EXAMPLE OF AWESOME | 😊 | 😢 | 💩 | EXAMPLE OF CRAPPY                                                                                                                  | NOTES + COMMENTS |
|--------------------|---------------------------------------------------------------------------------------------|--------------------|---|---|---|------------------------------------------------------------------------------------------------------------------------------------|------------------|
| AREA               | EXAMPLE OF CRAPPY                                                                           |                    |   |   |   |                                                                                                                                    |                  |
| DELIVERING VALUE   | WE DELIVER BETTER THINGS WE'RE PROUD OF IT AND STAKEHOLDERS ARE REALLY HAPPY                |                    |   |   |   | WE DELIVER CRAP. WE FEEL SHAMED TO DELIVER IT. OUR STAKEHOLDERS HATE US                                                            |                  |
| EASY TO RELEASE    | RELEASED IT SIMPLE, DATE PAUSELESS + MOSTLY AUTOMATED                                       |                    |   |   |   | RELEASED IT KILLED PAINFUL, LOTS OF MANUAL WORK, AND TAKE FOREVER                                                                  |                  |
| FUN                | WE LOVE BEING AT WORK, AND HAVE GREAT FUN WORKING TOGETHER                                  |                    |   |   |   | BORING                                                                                                                             |                  |
| HEALTH OF CODEBASE | WE'RE PROUD OF THE QUALITY OF OUR CODE. IT'S CLEAR, EASY TO READ, AND HAS A LOT OF COVERAGE |                    |   |   |   | OUR CODE IS A MESS OF PUNS, AND TECHNICAL WORDS IS ROLLING OUT OF CONTROL                                                          |                  |
| LEARNING           | WE'RE LEARNING LITTLE OF INTERESTING STUFF ALL THE TIME                                     |                    |   |   |   | WE NEVER HAVE TIME TO LEARN ANYTHING                                                                                               |                  |
| MISSION            | WE'RE EXACTLY WHAT WE ARE EXPECTED ABOUT IT!                                                |                    |   |   |   | WE HAVE NO IDEA WHY WE ARE HERE, THERE IT IS HIGH LEVEL PICTURE OF HOW OUR SO-CALLED MISSION IS COMPLETELY UNCLEAR AND UNINSPIRING |                  |
| PAWNS OR PLAYERS   | WE ARE IN CONTROL OF OUR SETTING AND WE DECIDE WHAT TO BUILD OR HOW TO BUILD IT             |                    |   |   |   | WE ARE JUST PLAIN IN A GAME OF CHESS WITH NO INFLUENCE OVER WHAT WE BUILD OR HOW WE BUILD IT                                       |                  |
| SPEED              | WE GET STUFF DONE QUICKLY AND MAINTAIN HIGH STANDARDS                                       |                    |   |   |   | WE NEVER TRY TO GET ANYTHING DONE. WE KEEP GETTING STUCK OR UNPREDICTABLE. STUCK KEEP GETTING STUCK OR UNPREDICTABLE               |                  |
| SUITABLE PROCESS   | OUR WAY OF WORKING FITS US PERFECTLY                                                        |                    |   |   |   | OUR WAY OF WORKING SUCKS                                                                                                           |                  |
| SUPPORT            | WE ALREADY GET GREAT SUPPORT + HELP FROM OUR TEAM FOR IT (FROM OTHER TEAMS)                 |                    |   |   |   | WE KEEP GETTING STUCK BECAUSE WE CAN'T GET THE SUPPORT + HELP THAT WE ASK FOR                                                      |                  |
| TEAMWORK           | WE ARE A TOTALLY GELLED SUPER-TEAM WITH AWESOME                                             |                    |   |   |   | WE ARE A BUNCH OF INDIVIDUALS THAT REGRET NOT TALKING AND WORKING WITH OTHER PEOPLE                                                |                  |

- Understanding the different groups and roles at Fiscal

- Finance, Product, Customer, Sales, Marketing.
- BDM v RDM v CAM v ADD.

- PoC ERP connector (D365, SAP or Oracle?)

- [Onboarding data fields](#)

- Minimum data fields: Some sort of unique ID, Invoice number, Invoice amount, Base amount, Invoice date, Entered date, Supplier ref, Supplier name
- Getting trial access to an ERP instance
- What are the development options and costs?
- What other data, e.g. invoice line items, would be useful?
- Would this help us onboard customers quicker?
- Should we work with a customer to get real data?

- Hackathons

- Does the NXG Wiki need updating, pruning, enhancing, etc?

- Learning from experimentation and failure

- As a high performing team, we need to be doing our BAU jobs day to a excellent standard, but also challenging ourselves and teams at the same time to experiment more in order for us to progress and get better. It is inevitable that some of these experiments will FAIL.
- Think about examples where you have experimented and failed. What did you learn and will be doing differently moving forward?
- Classic Quotes:
  - "When looking to innovate—fail forward. Innovation can be a chaotic process, embrace it and learn from your failures. If you're not failing every now and again, it's a sign you're not doing anything very innovative." —Scott D. Anthony
  - "I wouldn't be where I am now if I didn't fail, a lot. The good, the bad, it's all part of the success equation." —Mark Cuban
  - "If you think you can do a thing or think you can't do a thing, you're right." —Henry Ford
  - "No matter what your current condition, how or where you grew up, or what education or training you feel you lack, you can be successful in your chosen endeavor. It is spirit, fortitude, and hardiness that matter more than where you start." —Jack Ma
  - "The most dangerous poison is the feeling of achievement. The antidote is to every evening, think what can be done better tomorrow." —Ingvar Kamprad
  - "Most people give up just when they're about to achieve success. They quit on the one yard line. They give up at the last minute of the game, one foot from a winning touchdown." —H. Ross Perot
  - "It's fine to celebrate success but it is more important to heed the lessons of failure." —Bill Gates
  - "Develop success from failures. Discouragement and failure are two of the surest stepping stones to success." —Dale Carnegie
  - "I just love it when people say I can't do it, there's nothing that makes me feel better because all my life, people have said that I wasn't going to make it." —Ted Turner
  - "Failure is not the outcome—failure is not trying. Don't be afraid to fail." —Sara Blakely

## Questions

- Should anyone outside of Engineering be invited? e.g. Product, IT.

- Yes to the POs (to add insights).
- Should the day be fully remote, fully in person or does hybrid work?
  - Fully in person.

## Summary

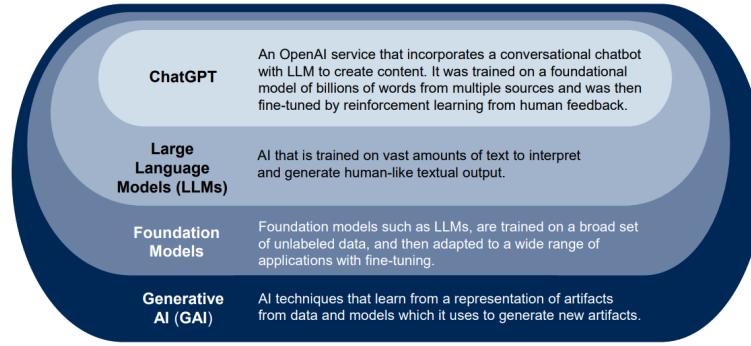
The purpose of this page is to discuss ML and AI opportunities for Fiscal.

## Context

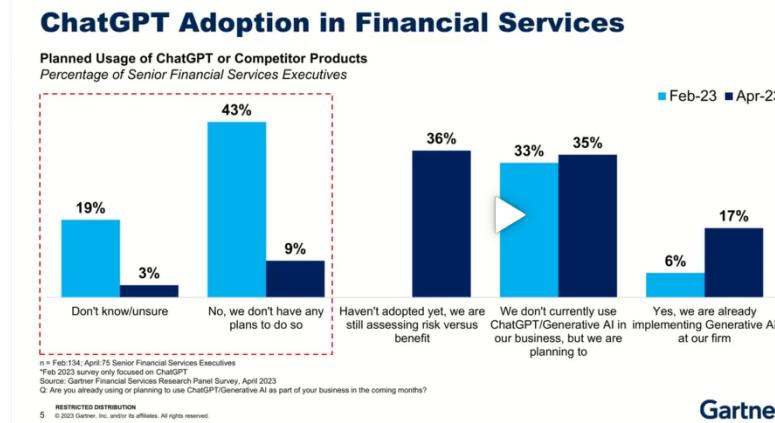
Our competitors' marketing machines are claiming ML/AI features, so we need to play the game too.

Launch-phobia is understandable, but with a fast-moving environment like AI it is generally a bad thing to wait for the "perfect" idea.

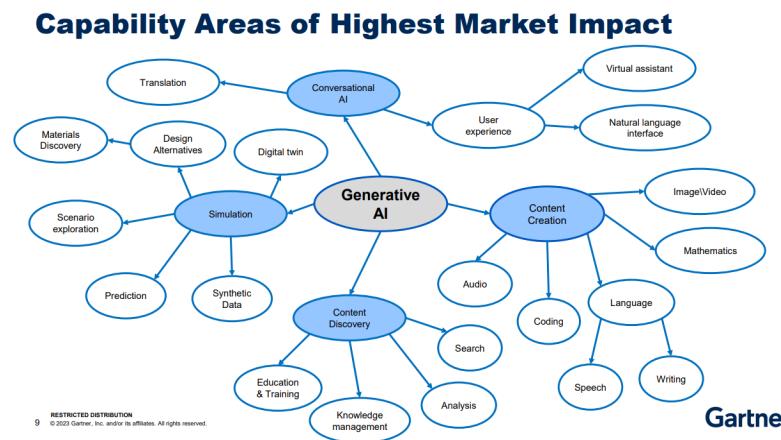
Remember that ChatGPT is a single solution from a single vendor (Microsoft), and Generative AI (GenAI) is part of a much broader category of technology.



ChatGPT adoption in financial services:



Capability areas of highest market impact:



Recommendations (Fiscal is still "raising awareness"):

| Raise awareness                                                                                                                                                                                                             | Build the capabilities you need                                                                                                                                                                                                     | Evaluate Use Cases & Get to work                                                                                                                                                                                                           |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• Increase understanding of C-Suite</li> <li>• Socialize GenAI with employees (pros/cons)</li> <li>• Conduct an opportunity/risk assessment and start assessing use cases</li> </ul> | <ul style="list-style-type: none"> <li>• Identify who will lead this work</li> <li>• Equip them for success – including understanding of expectations and value</li> <li>• Fill staffing gaps in IT and model management</li> </ul> | <ul style="list-style-type: none"> <li>• Identify where your organization should start &amp; choose use cases for success</li> <li>• Start POCs with some easy wins</li> <li>• Establish appropriate governance &amp; oversight</li> </ul> |

Understanding GenAI's intricacies is the first step in an organisation's journey. It is followed by steps like:

- Identify use cases that align with procurement processes.
- Prioritise based on value, feasibility and associated risks.
- Develop the team's digital and data literacy.
- Design and implement, embracing an iterative approach for continuous improvement.

## Notes

From Dan E:

- *The hype around ML/AI (and its greatest successes) are from scenarios where there are vast quantities of free training data ready to feed into the training process or there's no right answer so success/failure is a fuzzy question. Neither of those directly apply to Fiscal.*
- *So AI is a completely nebulous term and a lot of machine learning used in the real world is actually simple. I call it "self-configuring" i.e. products that take a statistical overview of the data provided by the customer and adjusts what it does according to that picture. We do that in lots of ways, e.g. when we look at your spending patterns for each supplier and use that knowledge to detect unusual payments, or when we look at the structure of invoice numbers issued by each supplier so we can flag up ones that don't fit that pattern, or when we build a heatmap of the times of day or days of the week when you do the most payments and we can flag up payments made at unusual times, and so on.*
- *The data science team over the last couple of years did a research project in conjunction with a uni (Craig will have the details) to build a machine learning model for a specific step of processing, where we take the individual scorer values and combine them into one number. The idea was to train a model to perform that operation by feeding it the actual user feedback obtained from NXG users.*
- *I'm skeptical about that being able to outperform a more "designed" approach, because as a fraction of the invoices we receive, the real risks are tiny, and we only show users a subset of those, and then the rest of the data set is noise, much of which will closely resemble real risks (very often the difference between a true and false positive isn't even captured in the data we receive!) So, there's a danger of "curve fitting" getting out of control. Also it's good to be able to explain what caused a risk to be highlighted, "why are you showing me this?" and that is much easier if you've consciously designed the risk model rather than it being a mysterious black box that just spits out answers. So right now the data science team is back to focussing more on that consciously-designed approach.*

## Resources

- [Ex-Google Officer Finally Speaks Out On The Dangers Of AI! - Mo Gawdat](#)

## Questions

- Which parts of NXG can we claim as already using ML/AI?
- How are our competitors, in particular Xelix, using ML/AI? Scorers? Chatbot? How do we find out more?
- How do companies like Esker talk about AI with their customers?
- Which parts of NXG would be best suited for ML? e.g. trend detection like our current transaction spikes

**Status:** Draft

This page documents the policy around permissions and access to NXG live environments in Azure, split into 3 roles:

- Developer
- Contributor
- Owner

## Developer

As part of their duties, developers are expected to:

- Diagnose and fix customer issues which cannot be resolved by support, as part of a rotation in app support.
- Supervising the deployment of changes into the live environment.
- Configuring SSO integration for a customer.
- Understanding when changes made would result in a cost increase in Azure services

## Azure Permissions

Live issues are by definition unexpected, and so it is not possible to know precisely what access will be required to diagnose and resolve an issue. For this reason Software Engineers are granted the `Monitoring Reader` role in all NXG live subscriptions. This allows access to a wide range of diagnostic information, including, but not limited to:

- **The ability to check the existence and basic properties of all Azure resources**, e.g. for diagnosing why a terraform change is failing to apply, validating that a build has been deployed, validating that a resource has been updated as expected etc...
- **The ability to view current and past Azure costs**, e.g. to anticipate expected Azure cost changes resulting from a change.
- **The ability to view resource use of SQL databases**, e.g. to understand if SQL database resource use is causing poor application performance, and to pinpoint which databases / customers are the cause of that resource use.
- **The ability to view resource use of the processor Azure container instance**, e.g. to understand if resource constraints are the root cause of file processing errors.
- **The ability to view Microsoft Defender for Cloud Scores**, e.g. for completing a task to implement recommendation and validate that recommendation has been applied correctly to the live environment.
- **The ability to view SQL Server Audit Logs**, e.g. to identify poorly performing SQL queries, or to identify if a given SQL query has been recently run for a given customer.
- **The ability to view App Insights telemetry and Application logs**, e.g. to understand what actions a user may have performed in the product to reproduce an issue, or to view exception details.
- **The ability to view assigned permissions for managed identities**, e.g. to diagnose an exception relating to Azure permissions.
- **The ability to view a list of containers in Azure Container Registries**, e.g. to diagnose an image pull error.

These permissions **exclude**:

- The ability to view secrets in Azure Key Vault.
- The ability to view the contents of storage accounts.

## B2C Permissions

As part of their role Software Engineers may be expected to setup or modify an SSO integration for a customer, which currently involves manual changes in Azure B2C as APIs do not yet exist to support completing this task automatically.

TBC - What permissions are needed for this?

## Database Permissions

Engineers also have the ability to automatically obtain read-only access to individual live databases, e.g. to investigate a foreign key constraint violation, or to validate that the deployed schema matches the expected state.

## NXG Permissions

Engineers have the ability to create Blank and Demo customers in the cloud tool, e.g. to perform post-release testing and to attempt to reproduce issues which cannot be reproduced in non-production environments.

## Hangfire

Engineers have access to the hangfire dashboard, which shows information about hangfire jobs, e.g. to view details of a failed Hangfire job. This dashboard also provides the ability to delete a job, which may be required to cancel a stuck job.

## Solr

Engineers have access to Solr, which provides information about the health of Solr, e.g. to investigate failed calls by NXG to Solr.

## Auth0

TODO

## CloudFlare

TODO

## Contributor

NXG also requires Engineers capable of completing tasks which require additional permissions than those granted by the Developer role, however are performed less frequently, e.g. (a non-exhaustive list)

- Restarting services

- Backing up and restoring databases
- Running ad-hoc scripts which modify data in live databases
- Running NXG commands which require access to keyvault
- Performing terraform state manipulation
- Correcting unexpected errors which may require arbitrary changes to Azure resources

## Azure Permissions

On each live NXG subscription Developers are assigned the `Contributor` Azure role.

## Owner

---

Azure recommend that Azure Subscriptions have 3 owners for the purpose of account recovery, however it is not expected that the owner account would be used except in extremely unusual circumstances.

## Summary

Fiscal have selected [Vitally](#) as its Customer Success Platform (CSP). Data will be obtained from NXG, Zendesk, Outlook, Salesforce and Xero.

This page focuses on the data that will be obtained from NXG.

## Integrations

Vitally supports multiple [integrations](#) for syncing data into Vitally on a recurring basis. These are the options for the NXG data:

- NXG copies the data to CSV files in **Amazon S3**, and Vitally periodically **pulls** the data.
- NXG stores data in **Postgres** tables, and Vitally periodically **pulls** the data.
- NXG periodically **pushes** data to Vitally using the **Vitally REST API**.

These are all do-able. The NXG data resides (or will reside) in [Biblioteca](#).

Although more work (and a degree of tie-in to Vitally), the team leads have a preference for using the Vitally REST API as:

- Fiscal don't have any form of AWS presence, so there is a fairly hefty startup cost to make sure we have the right compliance, permissions, auditing, etc.
- Fiscal don't use Postgres in Azure.
- The API offers the most flexible approach.

## Examples of NXG data

Two of the goals for the CSP are:

- Enable the CS team to handle more customers per person.
- Provide a trackable set of onboarding steps to our customers.

Examples of useful NXG data include:

- When people logged in for the first time.
- When people did something useful, like uploaded a file.
- When the first duplicate was found by a customer. Or should this be the first REAL duplicate?

Dan E has confirmed that these are things that could be distilled from Biblioteca. It has a table containing every uploaded file from every unit of every customer ever, so a query to find the min date by unit/file-type would be a start. Similar for duplicates (presumably this means the user confirms it's a real risk for the first time).

Dan E has also confirmed that we plan to have lots of points in the UI where we log actions taken by users specifically for this kind of tracking.

Jamie M has confirmed that we were/are planning on pushing Azure App Insights data into Biblioteca too, so possible easy win with anything that needs this.

## Questions

- Do we have a common identifier across NXG, Zendesk, Outlook, Salesforce and Xero?
- How do we obtain data from Xero? [Zapier](#) or via Salesforce or via Vitally REST API (if Xero supports GET requests).

## Introduction

This page describes the release process for deploying NXG Forensics upgrades to the live production environment.

Each production release will have a corresponding Release user story with the `Release` tag.

## Sprint Release Process

At the end of each sprint, all completed stories are deployed to the production environment. To do this all approved changes are merged to the main release branch in preparation for the deployment. Before this merge can take place a release readiness meeting is held to confirm that the changes are ready for release.

### Release Readiness Meeting

The release readiness meeting is held by the Change Advisory Board (CAB) to determine whether the proposed build is fit for release. The CAB consists of the Head of Operations, the Head of Software Development, Product Owners and senior members of the development team. Additional attendees may be required depending on the content of the release.

The main focus of the release readiness meeting will be to review the pre-release checklist and determine whether to approve the release. In the event that one of the items is not met, the group will evaluate the risk to determine whether to accept the issue or identify the corrective action that should be taken.

At the end of the meeting, providing the release has been approved, the release schedule will be agreed.

Minutes of the meeting are recorded in the sprint's deployment story in Azure DevOps.

### Pre-release Checklist

The following questions will be considered during the release readiness meeting:

- Have any changes that would impact FISCAL's legal obligations or compliance requirements (e.g. GDPR or ISO-27001)?
- Have any changes that would impact security been reviewed and have any risks been addressed or mitigated?
- Have any changes that would impact our Disaster Recovery or Business Continuity plans?
- Have we added or removed any additional packages to this release?
- Has the Azure Security Centre score been reduced?
- Does the security vulnerability scan indicate that any new vulnerabilities have been introduced?
- Have all unit tests completed successfully?
- Have all of the automated Selenium tests completed successfully?
- Has the solutions performance been impacted?
- Are all manual deployment tasks recorded?
- Has all required documentation been updated, including "What's new" text, Onboarding and security documentation?
- Are there any changes to the solutions ongoing running costs?
- Are there any changes the organisation needs to be aware of prior the release?
- Are there any other concerns or potential Risks?

### Release Approval

For the release to be approved the following members of the CAB must be in agreement:

- Head of Operations
- Head of Software Development
- At minimum 1 member of the Product Owners team
- Majority approval from members of the Senior Development team

If either Head of Operations or Head of Software Development are absence, they can delegate their authorisation to the other or to a member of the Senior Development team.

## Ad-hoc Release Process

On occasion, small ad-hoc releases may be required between sprint releases. These primarily will be required to rectify critical and high priority issues. In these cases only the required changes are "cherry-picked" from the development branch and merged to the main release branch, which is reviewed and approved prior to deployment.

Under these circumstances to expedite the release, the deployment can be approved by either the Head of Operations or Head of Software Development, a Product Owner and a member of the Senior Development team, without the need for the full release readiness meeting. A communication to the above parties should occur to ensure all are aware of the Ad-hoc release, the party that signed it off and logging the event.

The release should still be accompanied by a corresponding Release user story with the `Release` tag.

## Deployment

Once the release has been approved, it is deployed via the CI/CD pipeline. An Engineer will be oversee the automated release process to ensure it is successful and approve key stages as required, including infrastructure changes and updates to the deployment rings. A staggered approach is taken to deploying the rings to minimise the impact on the customer base in the event of an issue being discovered.

Once the release has completed, final checks will be performed on production to ensure that solution is behaving as expected. These include a manual review of the system health and executing the suite of automated Selenium tests.

Deployment takes place outside of core business hours for each region to minimise disruption to customers. Hotfixes may be the exception to this rule, where they are critical.

A new non-clustered, non-unique index added for the purpose of performance tuning is considered a standard change and can be approved through the following process.

The goal of this process is to make it easier to test indexes when performance tuning without having to apply the index to all customer databases before getting feedback.

## Process

1. A release work item should be created detailing the index to be added and the databases(s) to which the index will be added.
2. The index should be prefixed with `Temp_` to clearly identify it as a manually added index.
3. The index must be approved by at least two engineers (for example Maggie Duggan and Justin Pealing).
4. The index can then be applied manually to the target databases using access granted via PIMS.
5. A follow-up task must be created to remove the index when it is no longer required (for example because an equivalent index has been incorporated into the product).

## Notification of changes

A message with a link to the work item should be posted in the "Ad-hoc Releases" chat in MS teams prior to the index being applied. The purpose of this message is to serve as a notification to relevant parties that a change is about to be deployed. Where possible ~1 hrs notice should be given before deploying.

## Review of changes

Changes will be reviewed every sprint at the end of the release readiness meeting.

In cases where a customer is impacted by a live issue the following process may be followed to release a hotfix. Examples of issues where this would apply:

- Files stuck unable to process
- Issues logging in or accessing NXG (e.g. timeouts, poor API performance)
- Bugs causing NXG to display no data, or to display incorrect data
- Errors preventing users from applying workflow

## Process

### Tracking of changes

A release work item tagged with the `Hotfix` tag should be raised in the current sprint with details of the change, this work item should be updated with relevant details when the change is applied:

- Information about the customer issue that the hotfix is intended to address.
- A link to the corresponding pipeline run, and the PR that introduced the changes (if applicable)
- Who approved the change
- Information about how the change was tested
- Rationale as to why the change can be considered safe to release without impact to customers.

The purpose of this work item is to allow engineers to look up details of previous changes, and to provide evidence for auditing purposes.

### Notification of changes

A message with a link to the work item should be posted in the "Ad-hoc Releases" chat in MS teams prior to the release being applied. The purpose of this message is to serve as a notification to relevant parties that a change is about to be deployed. Where possible ~1 hrs notice should be given before deploying.

### Approval of changes

Changes can be approved by a Product Owner and 2 members of the engineering leadership team (Adam Awan, Daniel Earwicker, Jamie Mulcahy, Justin Pealing).

### Review of changes

Changes will be reviewed every sprint at the end of the release readiness meeting.

## Exclusions

- Permission changes must be approved as per the standard approval process for permission changes. Once the permission changes have been approved this process may be used to apply those changes by recording in the deployment work item who approved the permission changes.
- Cost increases must be approved separately.

A terraform change deployed via the terraform pipelines in Azure DevOps is considered a standard change, and can be approved through the following process:

## Process

### Preparation of changes

- The change should be applied and tested in hallway (with the exception of changes in `terraform-initial`, where testing in hallway is not applicable).
- At least two engineers must review the pull request.
- The change should be reviewed to ensure it will either not impact customers, or will be deployed out of hours.

### Tracking of changes

A release work item tagged with `Terraform Release` should be raised in the current sprint with details of the change, this work item should be updated with relevant details when the change is applied:

- A link to the corresponding pipeline run, and the PR that introduced the changes (if applicable)
- Who approved the change
- Information about how the change was tested
- Rationale as to why the change can be considered safe to release without impact to customers.

The purpose of this work item is to allow engineers to look up details of previous changes, and to provide evidence for auditing purposes.

### Notification of changes

A message with a link to the work item should be posted in the "Ad-hoc Releases" chat in MS teams prior to the release being applied. The purpose of this message is to serve as a notification to relevant parties that a change is about to be deployed. Where possible ~1 hrs notice should be given before deploying.

### Approval of changes

Terraform changes can be approved by 2 members of the engineering leadership team (Adam Awan, Daniel Earwicker, Jamie Mulcahy, Justin Pealing).

### Review of changes

Changes will be reviewed every sprint at the end of the release readiness meeting.

### Exclusions

- Permission changes must be approved as per the standard approval process for permission changes. Once the permission changes have been approved this process may be used to apply those changes by recording in the deployment work item who approved the permission changes.
- Cost increases must be approved separately.

### Example changes

- [Update shared queries](#)

## Random - in no way fixed - rota for running Retros

- Kevin
- Dylan
- Justin
- Craig
- *Richard*
- Vache
- James W
- Tom
- Alex
- Adam
- Andy
- Toni
- Mohi
- Dan
- Ryan
- Jamie

## Summary

The purpose of this page is to discuss and plan enhancements to Fiscal engineering in order to avoid bottlenecks before they appear and enabling significant growth. To keep an open mind to possible changes it would be beneficial to think on a factor of 10, i.e. imagine that Fiscal has 10 times more customers, or engineering 10 times more developers, or a customer has 10 times more units, 10 times more files, 10 times more transactions and so on.

## What is scalability?

By definition, scalability is the ability of a business or system to grow larger.

## Types

Resources usually fall into two categories:

- Horizontal or scale out, i.e., adding more resources.
- Vertical or scale up, i.e., improve existing resources.

## Dimensions

Scalability can be measured over multiple dimensions:

- Administrative scalability:
  - The ability for an increasing number of organizations or users to access a system.
  - *Fiscal*: What resources would we need to accommodate this extra capacity? (computationally increase, size of support team, amount of support tickets, performance of operations, etc). Running out of DBs.
- Functional scalability:
  - The ability to enhance the system by adding new functionality without disrupting existing activities.
  - *Fiscal*: What changes do we need to make to accommodate these extra functionalities?
- Geographic scalability:
  - The ability to maintain effectiveness during expansion from a local area to a larger region.
  - *Fiscal*: We currently have 3 regions (US, EU, UK). Will we expand to a fourth region in the future? How many customers can we hold inside a single region until we have to break it down into two regions? Total: US: 24/10/2023 - 39 customers (11 created after 01/01/2023) EU: 24/10/2023 - 179 customers (118 created after 01/01/2023) UK: 24/10/2023 - 240 customers (164 created after 01/01/2023) Standard & Implementation only: US: 24/10/2023 - 30 customers (6 created after 01/01/2023) EU: 24/10/2023 - 52 customers (4 created after 01/01/2023) UK: 24/10/2023 - 163 customers (94 created after 01/01/2023)
- Generation scalability:
  - The ability of a system to scale by adopting new generations of components.
  - *Fiscal*: AI? This could happen in many forms, GitHub copilot?
- Heterogeneous scalability:
  - The ability to adopt components from different vendors.
  - *Fiscal*: CreditSafe and LexisNexis to provide sanctions and credit monitoring. Is there potential new third parties? We showed interest on bank checking.
- Load scalability:
  - The ability for a distributed system to expand and contract to accommodate heavier or lighter loads, including, the ease with which a system or component can be modified, added, or removed, to accommodate changing loads.
  - *Fiscal*: This is probably the dimension where Engineering could have more impact. This section could cover many areas:
    - Database - MAXDOP
    - CPU
    - Caching
    - Asynchrony / Concurrency
    - Queuing
    - Networking - TCP port usage, redis on a private link
    - Code optimizations
    - Testing
    - Team working practices
    - Resilience - A failure of a component does not mean the failure of the entire system. Unfortunately we have seen a few of those where a single customer impacts a whole region.

## Software-Assurance-Maturity-Model-(SAMM).md

A Baseline assessment was carried out on the NXG Team by @<286F3CAF-93A4-4A66-9638-A3D10C7D4C5B> on 21/09/08 usign the following toolkit

### SAMM Assessment Toolbox v2.0.xlsx

General description for the 3 maturity levels

- The first maturity level aims at achieving an ad-hoc, best effort implementation of a particular activity within a stream.
- The second maturity level aims to establish a consistent, repeatable process that can be relied on.
- The third maturity level aims to maximize effectiveness through continuous improvement, based on effective and timely feedback.

## Key Findings

This is our maturity rating on a scale of 0-3

| Functions      | Current |
|----------------|---------|
| Governance     | 1.17    |
| Design         | 1.75    |
| Implementation | 1.42    |
| Verification   | 1.58    |
| Operations     | 1.88    |

Split by sub areas

![image.png](/..attachments/image-4e4d9fe1-f136-4556-9dc9-65667a839045.png =35% x)

## Phases

### Phase 1 (1st Jan 2021)

Goal: reach a 1.5 across all areas with the exception of governance as they will fall somewhat into Lesley's domain and her availability

| Business Functions | Current |
|--------------------|---------|
| Governance         | 1.42    |
| Design             | 2.08    |
| Implementation     | 1.71    |
| Verification       | 1.58    |
| Operations         | 2.04    |

## How

### Measure

### Goal

Did you define Key Performace Indicators (KPI) from available application security metrics?

Yes, for at least half of the metrics

Do you update the Application Security strategy and roadmap based on application security metrics and KPIs?

Yes, but review is ad-hoc

Have you identified a Security Champion for each development team?

Yes, for most or all of the teams

Do you classify applications according to business risk based on a simple and predefined set of questions?

Yes, most or all of them

Do you use a standard requirements framework to streamline the elicitation of security requirements?

Yes, for most or all of the applications

Do you enforce automated security checks in your build processes?

Yes, for at least half of the application

Do you prevent build of software if it's affected by vulnerabilities in dependencies?

Yes, for some applications

Do you track all known security defects in accessible locations?

Yes, for at least half of the applications

Do you enforce SLAs for fixing security defects?

Yes, for some applications

Do you follow a documented process for incident detection?

Yes, for most or all of the applications

### Phase 2 TBD

## Introduction

This page details the software development process currently used by FISCAL Technologies Software Development team.

## Agile Software Development Approach

The Software Development team adopted an Agile approach when they started the development of NXG Forensics. This was identified as the most appropriate approach for developing a modern SaaS software application.

In practice this means that the team follows the Scrum framework, relies heavily on automated testing and follows Continuous Integration and Continuous Delivery models. This results in software being developed and delivered on an iterative basis with updates being released to customers every two weeks.

## Software Development Lifecycle

Development is ongoing with frequent regular releases rather than a more traditional Waterfall approach with infrequent releases on an ad-hoc schedule.

The Software Development team follows the Scrum framework and this process is as follows:

- A short-term Product Backlog of features based on the current Product Roadmap is created by the Product Team. This is a list of User Stories which describe the feature to be developed and is a living document that can be modified based on the changing priorities of the business.
- Prior to the start of the Sprint, the team selects User Stories from the Product Backlog based on priority, breaks them down into tasks and sizes them. Based on these sizes and their previous velocity the team identifies the amount of work that can be delivered during the Sprint and moves it into the Sprint Backlog.
- During the Sprint the team designs, develops, tests and reviews the items in the Sprint Backlog ensuring that only fully completed tasks are released.
- On the final day of the Sprint the finished Stories are demonstrated to the rest of the organisation during the Sprint Showcase to provide an opportunity to gather additional feedback.
- After Showcase the holds a Sprint Retrospective, where they review the Sprint and identify any processes or behaviours that could be modified to improve the next Sprint.
- Finally the software is deployed to the live production environment.

This approach allows the team to deliver frequent updates that are constantly providing incremental value to FISCAL's customers.

## Verification & Release Strategy

The team has adopted a test strategy which relies heavily on automated test frameworks and is mainly focused around ensuring that a comprehensive set of unit tests exist across the product. This is supplemented by end-to-end tests undertaken by a Selenium automated test harness and manually by the team. Testing takes place throughout development and process is as follows:

- Developers create unit tests as part of each task they complete.
- Once the task is complete, the developer will ensure that all of the unit tests (including the additional tests created) are run successfully and that their software meets the acceptance criteria associated with the task. At this point the task is complete it is submitted for review.
- The changes (including associated tests) are reviewed by a second developer who may recommend modifications prior to approval. When the second developer is comfortable that the changes and any rework is complete, the modifications are approved, and the code is committed to the staging server.
- The unit tests are rerun as part of the merge and build process on staging server and must complete successfully before the new version is deployed.
- After the new version is deployed to the staging server, the developer will check their changes are working as expected. Additionally, automated performance tests executed either periodically or whenever a change is made that is likely to have a direct impact the product's performance.
- The Selenium automated tests are run nightly to provide additional confidence to catch any issues prior to release.
- Once deployed to the staging server, the Product Owner will review the changes to ensure that acceptance criteria are met and the User Story is ready for release.
- Prior to deployment, a platform change control review is held to ensure that the development, IT/security and product management teams are all comfortable with the updates that have been made to the software and that any outstanding risks have been discussed and mitigated as required.
- Finally, after updates have been deployed to the live production environment a light-weight sanity test is manually performed to confirm deployment has been successful.

## Customer Data

Customer data must not be used during development and verification of new features and must not be uploaded to development and test environments.

There is one exception to this rule; customer data may, with the customer's permission, be used for performance testing. In this case the data is held in a restricted performance test environment which is separate from the standard development and test environments.

## Branch Management & Change Control

To ensure that FISCAL Technologies can track all changes to its products, all code and script changes are held in a Git repository. Whenever a developer is starting on a new task, they create a separate branch based on the current version of Trunk. They will make all of their changes in this branch and these won't be committed to the main Trunk until then have been reviewed and approved by a second developer.

Product changes and modifications can only be released to live once they have been approved by either Chief Software Architect or System Architect.

## Package Management

To reduce maintenance, avoid introducing errors and make upgrades easier, FISCAL Technologies has a policy of using unmodified versions of third party software packages. Modified versions of open source packages should only be used in exceptional cases, when no other alternatives are available. In this event the change will go through the same review and approval process as any other code change so that it can be tracked. Additionally these changes should be pushed to the open source community for approval.

When a PR adds a new package (e.g. NuGet or NPM package) check the following and add the result to the PR, either as a comment or in the description:

- Check [Skyk Advisor](#) for indications that the package is unmaintained or insecure
- Check the license on [tl;dr Legal](#), e.g:
  - [MIT](#)
  - [Apache-2.0](#)
  - [ISC License](#)
  - [BSD 3-clause](#)
  - [BSD 2-clause](#)
  - [WTFPL](#)

## Summary

It is undeniable that effective metrics are essential for focusing attention and achieving results. However, it is also largely agreed that measuring software development in terms of individual developer productivity is not a good idea, as the problem is not one of productivity but rather one of team contribution.

We won't focus on metrics such as lines of code, bugs found, number of pull requests or test coverage, because these are prone to errors and can be easily gamed. Instead, we will focus on [DORA metrics](#), which are less gameable and have been proven to produce better software, faster.

Everyone at Fiscal should understand which metrics drive our software development, and what behaviors they encourage.

We will use OKRs to drive the visibility and improvement of our chosen metrics.

## DORA Metrics

*"DevOps Research and Assessment (DORA) is the largest and longest running research program of its kind, that seeks to understand the capabilities that drive software delivery and operations performance. DORA helps teams apply those capabilities, leading to better organisational performance."*

[DORA metrics](#) predict successful outcomes based on a model of what works and what does not. They drive improvement through experimentation and innovation.

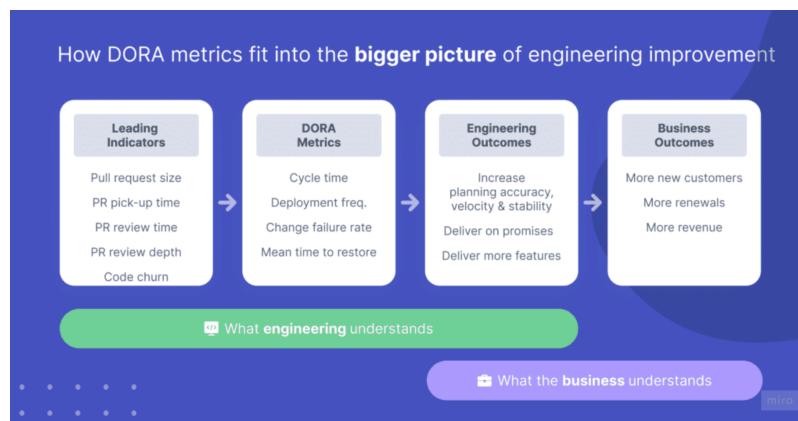
DORA metrics measure:

- The efficiency with which our process produces software, based on the throughput metrics (cycle time and deployment frequency).
- The quality of the software that we produce, based on the stability metrics (change failure rate and mean time to recovery).

Scoring well in all metrics makes a measurable difference to our chances of success.

### What are DORA metrics?

| Metric                                 | Definition                                                                                                        |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Cycle Time (AKA Lead Time for Changes) | Captures the time between a code change and when it is ready to be deployed to production.                        |
| Deployment Frequency                   | Refers to the frequency of successful software releases to production.                                            |
| Change Failure Rate                    | Indicates how often a team's changes or hotfixes lead to failures after the code has been deployed to production. |
| Mean Time to Recovery                  | Measures the time between an interruption due to deployment or system failure and full recovery.                  |



## DORA DevOps Quick Check

This consists of five multiple-choice questions and is used to compare performance to other companies. Justin and Paul provided the following answers on 19/09/2023:

- Lead time: one week to one month
- Deploy frequency: between once per week and once per month
- Change fail percentage: 16-30%
- Time to restore: less than one day
- Industry: Financial Services

We are performing better than 57% of 2022 *Accelerate State of DevOps* survey respondents.

## DORA Capabilities Analysis

This consists of multiple questions around [Continuous Integration](#), [Loosely Coupled Architecture](#) and [Generative Organisational Culture](#).

Justin and Paul completed the analysis on 19/09/2023, with the following results:

Continuous Integration: 4.2 / 5 Loosely Coupled Architecture: 3.3 / 5 Generative Organisational Culture: 3.8 / 5

This makes sense as:

- We have a relatively mature CI/CD pipeline.
- Although we are moving to a Modular Monolith our core functionality is still tightly coupled. We will need to reduce coupling if we significantly expand the size of the Engineering team.
- Using Westrum's organisational typology model, Fiscal are heading towards the Generative model, via a strong culture with good psychological safety.

## Other metrics

Some of the [SPACE metrics](#) could be of use to Fiscal but we will focus on the DORA metrics initially.

## Next Steps

- How do we measure, store and visualise our metrics?
- How do we automate the above?
- How do we ensure that the metrics are driving value vs gamification?
- How do we use the metrics to improve engineering outcomes?

## McKinsey Article On Developer Productivity

Consulting group McKinsey & Company have recently published an [article](#) which has been widely criticised. Grady Booch said “*It’s rubbish*”, Kent Beck said “*The report is so absurd and naive that it makes no sense to critique it in detail*” and Daniel Terhorst-North said, “*Don’t try to measure the individual contribution of a unit in a complex adaptive system, because the premise of the question is flawed*”.

## Other Links

- [What Makes a Great Team at Google](#)
- [The Worst Programmer I know](#)
- [My Response To The NONSENSE McKinsey Article On Developer Productivity](#)
- [Accelerate: The Science of Lean Software and Devops: Building and Scaling High Performing Technology Organizations](#)

The following items are implicit acceptance criteria that apply to all stories and must be met before the story can be approved.

## Security

- All passwords should be stored and processed securely
- No part of the solution should use a commonly known password
- Customer data must be segregated, and the same goes for unit data within a customer. Where data is pooled from multiple such contexts, e.g. the reports DB stores data from multiple units in one table, extra care must be taken to filter queries appropriately to match the user's access to the units.
- Audit logs should be entered before the action in case of partial failure

## User Interface

- All changes should work in the following support browsers:
  - Google Chrome
  - Mozilla Firefox
  - Microsoft Edge
- The UI should render updates while retrieving data in the background
- The UI should be a displayable on a minimum screen resolution of **1366 by 768**

## Quality

- Every story and task needs to be tested
- Appropriate and meaningful automated tests should be developed for every story
- New or updated code should pass all existing tests
- Unit tests should cover the range of cases enabled by the code, e.g. if a method takes an enum with three possible values, that implies there should be (at least) three test to cover those cases.
- If certain patterns must be followed that cannot be detected by the compiler, use unit tests and reflection to assert those patterns, e.g.
  - classes matching a certain criteria must have some attribute
  - for each member `M` of an enum, there must be a class named `WidgetM`

## Deployment Ready

- Completion of any individual story should leave the code in a potentially ship-able state

## Non-Breaking Changes

- No change to our solution should break an existing customer's configuration
- If a change is identified that would break the solution for a customer, then the customer should be notified in advance and advised as to how they can modify their configuration before the change is released

## **Team-Behaviours.md**

This section identifies and describes a number of key behaviours which the development team feel enables them to deliver the highest quality customer focused solutions possible. These behaviours encourage both engineering best practices and more importantly the philosophy behind the team's approach to ensure that we work together to deliver customer value in a collaborative and supportive way.

The following behaviours represent the team's values and each team member has a responsibility to follow them.

### **Communicate, Communicate, Communicate...**

- Ensure that you understand stories, tasks and their acceptance criteria and clarify any ambiguities
- Ask questions early
- Ask for help if blocked for more than 30 minutes
- Demo changes to Product Owner as you progress through the story

### **Maintaining a Collaborate & Supportive Environment**

- Focus on completing stories and achieving sprint goals
- Before selecting the next story see if you can help to complete in progress stories
- Completing stories is more important than starting new ones
- Provide guidance and help to other team members
- Ensure our colleagues and customers are getting the most out of our solutions

### **Deliver Customer Value**

- We want to deliver customer value on a frequent basis
- We need to ensure that the stories we deliver meet our customer needs
- Stories need to meet their acceptance criteria
- Wherever possible look for opportunities to delight the customer

### **Ensure High Quality**

- Each team member is responsible for the quality of the code they produce
- Every story and task needs to be tested
- Appropriate and meaningful automated tests should be developed for every story
- New or updated code should pass all existing tests
- Missing automated test coverage should be added whenever an existing area of code is updated or enhanced

### **Staying Security Focused**

- It is a team responsibility to ensure the security of the software they develop
- Security should be built into our solutions as we develop them
- As code is developed it is important not to introduce potential security vulnerabilities and issues
- Security concerns should be raised and discussed with the Chief Software Architect and the Systems Architect

### **Refactoring Frequently & Appropriately**

- It is important to refactor code to ensure that it can be effectively extended and it is kept clean and maintainable
- We should not refactor for the sake of refactoring
- Refactoring should be performed as it is required and incorporated into sprint plans
- Appropriate automated tests should be in place before the refactoring begins

### **Continuously Improving**

- We should be continuously looking for opportunities to improve our processes to optimise their efficiency and effectiveness
- This will allow us to eliminate waste ensuring that we are delivering the maximum amount of value to our customers in the shortest amount of time

### **Individual & Team Development**

- You should see research activities as an opportunity to develop and broaden your skill set
- You should look for opportunities to select tasks from the sprint backlog that will enable you to develop and build on your existing skill base
- You are encouraged to research and recommend new technologies which may benefit both the team and the solutions we deliver
- We should discuss and share ideas with other members of the team to help both your own and their development

Notes:

Select all that apply:

|                                                                                                                  |                                                                                   |
|------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Policies and procedures concerning change management for Application Development or Program Changes              | Policies and procedures concerning change management for Emergency Changes        |
| Policies and procedures concerning change management for acquisition and Implementation of new projects/hardware | Policies and procedures concerning change management for Configuration Management |
| <b>Secure coding practices, including third-party development</b>                                                | List of and business use of encryption tools                                      |
| Copy of certifications of proprietary encryption algorithms                                                      | Not Applicable                                                                    |

segementation of duty

graph TD; A[R&D] --> B[Peer Review] --> C[Through to development] --> D[Senior Review] --> E[CAB]; B --> A; E --> F[Release Sign Off] F --> G[Automated release of code requiring multiple sign offs]

The following privileges are reviewed and approved by the product CAB and Platform CAB due to impact to Support structure, additional privilege, and additional requirements to audit, monitor and report on the use and continued improvement of the security longer term.

- [Justification:](#)
- [The aim:](#)
- [Policy:](#)
  - [Key vault](#)
- [AKS](#)
  - [Roles: Platform Team – Read Access](#)
  - [Roles: Lead Developers - Read Access](#)
  - [Roles: Lead Developers and Platform Team – Write Access](#)
- [Blob Storage](#)
  - [Roles: Lead Developers and Platform Team – Read Access](#)
  - [Roles: Lead Developers – Contributor Access](#)
- [SQL](#)
  - [Roles: Database Administrators, Lead Developers and Platform Team – Read Access](#)
  - [Roles: Database Administrators, Lead Developers and Platform Team – Database Owner access to all Database Resources via nxg-live-admins group](#)
- [Break Glass – High Level Privileges](#)
  - [Roles: Lead Developers](#)
    - [In event of use:](#)

## **Justification:**

To enable additional access to the platform for escalation and support purposes. These permissions will act as escalation for the engineering teams to the team leads and Subject matter experts.

As such a set of permissions are required for the following key groups: Lead Developers, Platform Team, and Database Administrators.

## **The aim:**

- Provide escalation ability for the key groups to triage issues in our production environment.
- Make specific modifications to the production environment that we are currently unable to be made through the defined CICD pipelines as part of our release processes.
- Provide access where we do not currently have appropriate tooling to management the environment without privilege.

These permissions are primarily for use for deeper investigation into the production environment in event for break fix or where an event occurs where a break glass event occurs.

## **Policy:**

- The permissions must be reviewed at least annually or when material change occurs.
- Use of these permissions must be audited and reviewed at least monthly, preferably bi-weekly to ensure memory of access is specified, documented, and reported upon to ensure reasonable use.
  - Recommendation to review at the start of every DevOps meeting.
- Use of these permissions must be limited to the minimal scope possible to complete the task wherever possible.
- In event of accessing customer environments support cases must be raised, the access request noted with the incident ID and the customer informed (by proxy of the support ticket).
- In event of misuse, incidents must be escalated as a security incident via the helpdesk and the CISO and Director of IT and Security informed.
  - Misuse may result in revocation of the privileges and may be subject to disciplinary and / or legal action inline with the violation clauses of the companies governing policy.
  - Use outside of the defined use case could constitute misuse.
- Incidents must reviewed / managed following the ISMS Incident Management policies.
- Where repeat use of privilege occurs for particular use cases; an appropriate review should be scheduled to ensure the actions are appropriate or if additional processes / tooling is required.
- In event of material change to these permissions or requirements; A review must occur and request for change of access raised following the standard access control processes.
  - This policy must be updated inline with the changes approved by the associated CABs.
- A formal permission review will occur to ensure removal or addition of required permissions are actioned in line with compliance / security / operational best practices and business requirements.

Scope:

The required groups proposed for use in the permissions structure:

- Software Development – Lead Developers
- Software Development – Platform Team
- Database Administrators – Database Administrators

Access is implemented at NXG Management Group Level and all subordinate subscriptions unless specified:

## **Key vault**

###**Roles:** Lead Developers - Read and Amend access.

- **Justification:** Required to support production issues with service principals and customer environments.
- **Permissions:**
  - Key Vault Secrets User
    - Allows engineers to read secrets from key vaults.
    - Sensitivity: Very High
  - Key Vault Secrets Officer
    - Allows Engineers to perform any action on the secrets of a key vault, except manage permissions.
    - Sensitivity: Very High
- **Risks:**
  - As both permissions either provide secrets used in the product and can impersonate accounts or leverage access as another account, also access at a later date if keys are kept and not cycled this poses a significant risk.
- **Mitigations:**
  - Access to these areas must require a second person to sign off access.
  - Keys must be cycled frequently in line with ISMS policies.

## Roles: Platform Team - Read access.

- **Justification:** Required to support production issues with service principals and customer environments.
- **Permissions:**
  - Key Vault Secrets User
    - Allows engineers to read secrets from key vaults.
    - Sensitivity: Very High
- **Risks:**
  - As permissions provide secrets used in the product and can impersonate accounts or leverage access as another account, also access at a later date if keys are kept and not cycled this poses a significant risk.
- **Mitigations:**
  - Access to these areas must require a second person to sign off access.
  - Keys must be cycled frequently in line with ISMS policies.

## AKS

### Roles: Platform Team – Read Access

- **Justification:** Required for monitoring and management of the platform, troubleshooting, capacity management, performance tuning, etc.
  - As this is a core component of the platform teams working requirement permanent access has been requested with a review yearly.
- **Permissions:**
  - Azure Kubernetes Service Cluster User Role
    - Lists cluster user credential action.
    - Sensitivity: Low
  - Azure Kubernetes Service RBAC Reader
    - Allows read-only access to see most objects in a namespace. It does not allow viewing roles or role bindings. This role does not allow viewing Secrets, since reading the contents of Secrets enables access to Service Account credentials in the namespace, which would allow API access as any Service Account in the namespace (a form of privilege escalation). Applying this role at cluster scope will give access across all namespaces.
    - Sensitivity: Moderate
- **Risks:**
  - Permissions provides access to view structure, design, capacity, and operational elements related to AKS and could be weaponised for an attack.
  - Access would be permanently applied to the accounts.
- **Mitigations:**
  - The roles do not grant access to secrets that would potentially allow privileged escalation.
  - Informational alerts should be enabled but can be reviewed if they are required at a later date if they are adding limited to no posable value.

### Roles: Lead Developers - Read Access

- **Justification:** Required for support escalation, monitoring and management of product changes, troubleshooting, ML review and capacity management, performance tuning, etc.
  - Due to this not being a core requirement for the role but required for escalation or where dedicated stories / projects are enacted we recommend the ability to enable for a maximum of 2-week period to support the sprint development period but not have fulltime access.
- **Permissions:**
  - Azure Kubernetes Service Cluster User Role
    - Lists cluster user credential action.
    - Sensitivity: Low
  - Azure Kubernetes Service RBAC Reader
    - Allows read-only access to see most objects in a namespace. It does not allow viewing roles or role bindings. This role does not allow viewing Secrets, since reading the contents of Secrets enables access to Service Account credentials in the namespace, which would allow API access as any Service Account in the namespace (a form of privilege escalation). Applying this role at cluster scope will give access across all namespaces.
    - Sensitivity: Moderate
- **Risks:**
  - Permissions provides access to view structure, design, capacity, and operational elements related to AKS and could be weaponised for an attack.
  - Access would allow extended access to the privilege once applied to the accounts.
- **Mitigations:**
  - The roles do not grant access to secrets that would potentially allow privileged escalation.
  - Informational alerts should be enabled but can be reviewed if they are required at a later date if they are adding limited to no posable value.
  - Role should only be applied for the extended period when planned activities in this environment are scheduled as part of sprint.

### Roles: Lead Developers and Platform Team – Write Access

- **Justification:** Required for Manual changes where break fix is required, or an environmental change is required where CICD pipelines could have environment breaking effects.
- **Permissions:**
  - RBAC cluster administrator
    - Let's engineers manage all resources in the cluster.
  - Sensitivity: High
- **Risks:**
  - Permissions provides access to amend the infrastructure, capacity, and operational elements related to AKS and can be used to impact the availability of the environment, compromise environmental elements that hold data and could have impact to operating costs.
- **Mitigations:**
  - Access to these areas must require a second person to sign off access.
  - Where Implementation of manual changes must occur outside of the standard CI/CD pipelines; changes must be QA'd with a second person prior to implementation inline with the standard release process.
  - Where break fix occurs review of the implemented changes must occur following resolution to ensure changes are incorporated into the codebase / environment, or ongoing requirements are documented.

## Blob Storage

### Roles: Lead Developers and Platform Team – Read Access

- **Justification:** Required for review of logs and data when break fix is required.
- **Permissions:**
  - Storage Blob Data Reader
    - Allows for read access to Azure Storage blob containers and data.
    - Sensitivity: High – Depending on location
  - Storage Table Data Reader

- Allows for read access to Azure Storage tables and entities.
- Sensitivity: High – Depending on location
- Reader and Data Access
  - Allows access to list storage keys for storage accounts.
  - Sensitivity: High – Depending on location
- Risks:
  - Permissions provide access to uploaded customer data. Read only access can also allow exfiltration of the data and sensitivity of reviewing of logs.
  - As this access will also be granted at top level this will give access to the likes of terraform state and all storage at all regions.
  - Except for the customer data that holds our highest level of confidentiality and sensitivity as a business, though the access only gives access to read the information that can be gathered could potentially be weaponised for attack purposes.
- Mitigations:
  - Efforts to reduce duplicated customer data held in this space must be reviewed.
  - Access must be audited, alerted, and reported upon at minimum.

## Roles: Lead Developers – Contributor Access

- Justification: “Contributor for storage tables is needed to modify the global user tables when the UserID changes and Contributor for blobs is useful to clear out any invalid blobs (such as incorrect supplier blobs, due to bugs or incidents), and to upload blobs related to ML if any issues arise.” Though these events are uncommon currently in current events these events are escalated to Daniel Earwicker or Jamie Mulcahy for access currently, so aim is to widen access purely to lead developers.
- Permissions:
  - Storage Blob Data Contributor
    - Allows for read, write, and delete access to Azure Storage blob containers and data.
    - Sensitivity: High
  - Storage Table Data Contributor
    - Allows for read, write, and delete access to Azure Storage tables and entities.
    - Sensitivity: High
- Risks:
  - Primary concerns relate to access to uploaded customer data, ability to inject data and exfiltration of the data. Environment also provides of review of logs. As this access will also be granted at top level this will give access to the likes of terraform state and all storage at all regions.
  - With access to contribute this will allow destruction and manipulation of data potentially impacting its integrity or potentially masking attack.
  - Except for the customer data that holds our highest level of confidentiality and sensitivity as a business, access to the likes of terraform state could cause irrevocable damage to the product if maliciously modified including impact to costs, impact to customers, performance, deployment / distribution of services, etc.
- Mitigations:
  - Efforts to reduce duplicated customer data held in this space must be reviewed.
  - Log file locations must be reviewed to ensure they are secured from tampering.
  - There is an understanding that longer term there should be tooling to enable management of these activities without the need for such broad access, but currently there is no ETA on additional tooling.
  - Access to these areas must require a second person to sign off access.

## SQL

### Roles: Database Administrators, Lead Developers and Platform Team – Read Access

- Biblioteca – Region Manager
  - Justification: To review customer telemetry data
  - Sensitivity: Moderate
- Credit Monitoring – ABED
  - Justification: To review output of scorers, and issues reported with data.
  - Sensitivity: Low
- All NXG databases - live SQL diagnostics data
  - Justification: Required to support and reduce time to investigate and current process also provides access to customer data which shouldn't be required for a significant portion of support requests. Case already open and reviewed by Maggie Duggan for QA on access levels: <https://fiscalsupport.zendesk.com/agent/tickets/26795>
  - Recommendation provided by Maggie that they test the recommended permissions to review if any sensitive elements are found and deny inappropriate tables from the scopes prior to rollout.
  - Sensitivity: Moderate
  - Risks:
    - Limited due to restriction of areas allowed to view and allocation specified as read only. – Covered in primary request
  - Mitigations:
    - Access is restricted to specific tables and areas of the databases to minimise risk and access to sensitive data.

### Roles: Database Administrators, Lead Developers and Platform Team – Database Owner access to all Database Resources via nxg-live-admins group

- Justification: Break Glass in event of incident, some areas of the product activities still require full access currently to the database server e.g., Customer database clones, backups and data restores. Credit Monitoring – ABED requires full access for issue resolution at database level. Aim to remove need long term though tooling to allow write access to individual databases but currently no ETA for implementation.
- Permissions:
  - Delegated through the NXG-LIVE-ADMINS group.
  - Provides DBO level access to the database servers.
  - Sensitivity: Very High
- Risks:
  - Privilege grants full access to the database servers as Owner and allows administrative controls at database server level.
  - Privilege can be used to elevate privilege and make changes to databases and server configurations.
- Mitigations:
  - Aim to remove need long term though tooling to allow write access to individual databases but currently no ETA for implementation. Must be reviewed for requirement at annual review.
  - Secondary sign off is required prior to access and access must be audited, alerted, and reported upon.
  - In event of use, a session should be organised to review its use, identify areas of privilege requirement or mitigation steps going forward.
  - Exceptions to the requirement of a session include:
    - Database recovery
    - Cloning Databases for ERP Upgrades
    - Credit Monitoring – Abed issue resolution
    - Until tooling is developed and implemented the following are considered exceptions. Once tooling is in place privilege must not be used for:
      - Update to individual databases
      - Loading customer unit tables
      - Unblocking of blocked files
      - Unblock processing and fix customer states.

- Additions to the exceptions list must be reviewed at DevOps review and signed off by Lead Developers and Director IT and Security.

## Break Glass – High Level Privileges

### Roles: Lead Developers

- Justification: Both levels of access are considered break glass and should only be used in extreme measures where absolutely required.
  - All efforts should be made to minimise scope and ensure that understanding of required privilege are understood post any usage.
- Permissions:
  - Contributor – Globally
    - Sensitivity: Extreme
  - Owner – Globally
    - Sensitivity: Extreme
- Risks:
  - Privilege grants full access to the entire environment and the associated resources.
  - Owner allows ability to append privilege levels within the environment.
  - Privilege can be used to elevate privilege and make changes to configurations of all services.
  - Privilege can be used to manually configure and deploy resources outside of CI/CD Pipelines.
- Mitigations:
  - Secondary sign off is required prior to access and access must be audited, alerted, and reported upon.
  - Where Implementation of manual changes must occur outside of the standard CI/CD pipelines; changes must be QA'd with a second person prior to implementation in line with the standard release process.
  - Where break fix occurs review of the implemented changes must occur following resolution to ensure changes are incorporated into the codebase / environment, or ongoing requirements are documented.

### In event of use:

- a session must be organised to review its use, identify areas of privilege requirement or mitigation steps going forward.
  - No exceptions allowed.
    - Where an exception is identified this must be reviewed at DevOps review and signed off by Lead Developers and Director IT and Security.
    - Remediation of the exception must be scheduled for resolution at the earliest possible convenience.
  - Where changes in privilege requirements are identified they must be reviewed at DevOps and signed off by the Lead Developers, Product CAB, and Director of IT and Security.

The sub-pages here are a subset of the controls listed in **Annex A (normative)** at the end of INTERNATIONAL STANDARD ISO/IEC 27001, Third Edition 2022-10.

The plan is to annotate them with text and/or links to our guidance, to see how closely we are aligned, and so that information is "indexed" in a way that an auditor would find helpful.

Annotations look like this, and typically link to items under [Infrastructure and Patterns](#).

## Control

Information security should be integrated into project management.

## Purpose

- To ensure information security risks related to projects and deliverables are effectively addressed in project management throughout the project life cycle.

## Guidance

Information security should be integrated into project management to ensure information security risks are addressed as part of the project management. This can be applied to any type of project regardless of its complexity, size, duration, discipline or application area (e.g. a project for a core business process, ICT, facility management or other supporting processes).

The project management in use should require that:

1. information security risks are assessed and treated at an early stage and periodically as part of project risks throughout the project life cycle;
2. information security requirements [e.g. application security requirements (8.26), requirements for complying with intellectual property rights (5.32), etc.] are addressed in the early stages of projects;
3. information security risks associated with the execution of projects, such as security of internal and external communication aspects are considered and treated throughout the project life cycle;
4. progress on information security risk treatment is reviewed and effectiveness of the treatment is evaluated and tested.

The appropriateness of the information security considerations and activities should be followed up at predefined stages by suitable persons or governance bodies, such as the project steering committee.

Responsibilities and authorities for information security relevant to the project should be defined and allocated to specified roles.

Information security requirements for products or services to be delivered by the project should be determined using various methods, including deriving compliance requirements from information security policy, topic-specific policies and regulations. Further information security requirements can be derived from activities such as threat modelling, incident reviews, use of vulnerability thresholds or contingency planning, thus ensuring that the architecture and design of information systems are protected against known threats based on the operational environment.

Information security requirements should be determined for all types of projects, not only ICT development projects.

- The following should also be considered when determining these requirements:

1. what information is involved (information determination), what are the corresponding information security needs (classification; see 5.12) and the potential negative business impact which can result from lack of adequate security;
2. the required protection needs of information and other associated assets involved, particularly in terms of confidentiality, integrity and availability;
3. the level of confidence or assurance required towards the claimed identity of entities in order to derive the authentication requirements;
4. access provisioning and authorization processes, for customers and other potential business users as well as for privileged or technical users such as relevant project members, potential operation staff or external suppliers;
5. informing users of their duties and responsibilities;
6. requirements derived from business processes, such as transaction logging and monitoring, nonrepudiation requirements;
7. requirements mandated by other information security controls (e.g. interfaces to logging and monitoring or data leakage detection systems);
8. compliance with the legal, statutory, regulatory and contractual environment in which the organization operates;
9. level of confidence or assurance required for third parties to meet the organization's information security policy and topic-specific policies including relevant security clauses in any agreements or contracts.

## Control

Allocation and management of authentication information should be controlled by a management process, including advising personnel on the appropriate handling of authentication information.

## Purpose

To ensure proper entity authentication and prevent failures of authentication processes.

## Guidance

### Allocation of authentication information

The allocation and management process should ensure that:

- personal passwords or personal identification numbers (PINs) generated automatically during enrolment processes as temporary secret authentication information are non-guessable and unique for each person, and that users are required to change them after the first use;
- procedures are established to verify the identity of a user prior to providing new, replacement or temporary authentication information;
- authentication information, including temporary authentication information, is transmitted to users in a secure manner (e.g. over an authenticated and protected channel) and the use of unprotected (clear text) electronic mail messages for this purpose is avoided;
- users acknowledge receipt of authentication information;
- default authentication information as predefined or provided by vendors is changed immediately following installation of systems or software;
- records of significant events concerning allocation and management of authentication information are kept and their confidentiality is granted, and that the record-keeping method is approved (e.g. by using an approved password vault tool).

### User responsibilities

Any person having access to or using authentication information should be advised to ensure that:

- secret authentication information such as passwords are kept confidential. Personal secret authentication information is not to be shared with anyone. Secret authentication information used in the context of identities linked to multiple users or linked to non-personal entities are solely shared with authorized persons;
- affected or compromised authentication information is changed immediately upon notification of or any other indication of a compromise;
- when passwords are used as authentication information, strong passwords according to best practice recommendations are selected, for example:
  - 1. passwords are not based on anything somebody else can easily guess or obtain using person related information (e.g. names, telephone numbers and dates of birth);
  - 2. passwords are not based on dictionary words or combinations thereof;
  - 3. use easy to remember passphrases and try to include alphanumerical and special characters;
  - 4. passwords have a minimum length;
- the same passwords are not used across distinct services and systems;
- the obligation to follow these rules is also included in terms and conditions of employment (see 6.2).

### Password management system

When passwords are used as authentication information, the password management system should:

1. allow users to select and change their own passwords and include a confirmation procedure to address input errors;
2. enforce strong passwords according to good practice recommendations [see c) of "User responsibilities"];
3. force users to change their passwords at first login;
4. enforce password changes as necessary, for example after a security incident, or upon termination or change of employment when a user has known passwords for identities that remain active (e.g. shared identities);
5. prevent re-use of previous passwords;
6. prevent the use of commonly-used passwords and compromised usernames, password combinations from hacked systems;
7. not display passwords on the screen when being entered;
8. store and transmit passwords in protected form.

Password encryption and hashing should be performed according to approved cryptographic techniques for passwords (see 8.24).

### Other information

Passwords or passphrases are a commonly used type of authentication information and are a common means of verifying a user's identity. Other types of authentication information are cryptographic keys, data stored on hardware tokens (e.g. smart cards) that produce authentication codes and biometric data such as iris scans or fingerprints. Additional information can be found in the ISO/IEC 24760 series.

Requiring frequent change of passwords can be problematic because users can get annoyed by the frequent changes, forget new passwords, note them down in unsafe places, or choose unsafe passwords. Provision of single sign on (SSO) or other authentication management tools (e.g. password vaults) reduces the amount of authentication information that users are required to protect and can thereby increase the effectiveness of this control. However, these tools can also increase the impact of disclosure of authentication information.

Some applications require user passwords to be assigned by an independent authority. In such cases, a), c) and d) of "Password management system" do not apply.

## Control

Read and write access to source code, development tools and software libraries should be appropriately managed.

## Purpose

To prevent the introduction of unauthorized functionality, avoid unintentional or malicious changes and to maintain the confidentiality of valuable intellectual property.

## Guidance

Access to source code and associated items (such as designs, specifications, verification plans and validation plans) and development tools (e.g. compilers, builders, integration tools, test platforms and environments) should be strictly controlled.

For source code, this can be achieved by controlling central storage of such code, preferably in source code management system.

[Azure DevOps](#) stores our source code and controls all access to it.

Read access and write access to source code can differ based on the personnel's role. For example, read access to source code can be broadly provided inside the organization, but write access to source code is only made available to privileged personnel or designated owners.

[Azure DevOps](#) permissions are used to restrict access to authorised users only.

Where code components are used by several developers within an organization, read access to a centralized code repository should be implemented.

We follow the [monorepo](#) approach common in most organisations today, where all engineers all have read/write access to the same single code repository.

Furthermore, if open-source code or third-party code components are used inside an organization, read access to such external code repositories can be broadly provided. However, write access should still be restricted.

We reference pre-built [third-party Packages](#) in public registries, and are not responsible for copies of the source code of such packages.

The following guidelines should be considered to control access to program source libraries in order to reduce the potential for corruption of computer programs:

1. managing the access to program source code and the program source libraries according to established procedures;

[Azure DevOps](#) permissions are assigned according to each employee's role. 2. granting read and write access to source code based on business needs and managed to address risks of alteration or misuse and according to established procedures; Only employees working on the source code are granted access; any exceptions would be controlled by IT. 3. updating of source code and associated items and granting of access to source code in accordance with change control procedures (see 8.32) and only performing it after appropriate authorization has been received; This is controlled by our PR process. 4. not granting developers direct access to the source code repository, but through developer tools that control activities and authorizations on the source code; The repository is managed by [Azure DevOps](#) and all access is via the secure public APIS of that service. 5. holding program listings in a secure environment, where read and write access should be appropriately managed and assigned; The repository is managed by [Azure DevOps](#) and all access is via the secure public APIS of that service. 6. maintaining an audit log of all accesses and of all changes to source code. The [source repository](#) retains full history of all changes made to code.

If the program source code is intended to be published, additional controls to provide assurance on its integrity (e.g. digital signature) should be considered.

There are no plans to do this.

## Other information

If access to source code is not properly controlled, source code can be modified or some data in the development environment (e.g. copies of production data, configuration details) can be retrieved by unauthorized persons.

## Control

Secure authentication technologies and procedures should be implemented based on information access restrictions and the topic-specific policy on access control.

## Purpose

To ensure a user or an entity is securely authenticated, when access to systems, applications and services is granted.

## Guidance

GENERAL NOTE: Except where otherwise stated below:

- for our internal processes involving employees, these controls are implemented by [Azure Active Directory](#).
- for our customer-facing products that require authentication of end users, these controls are implemented by [external services](#) that perform all authentication of our end users on our behalf.

A suitable authentication technique should be chosen to substantiate the claimed identity of a user, software, messages and other entities.

The strength of authentication should be appropriate for the classification of the information to be accessed. Where strong authentication and identity verification is required, authentication methods alternative to passwords, such as digital certificates, smart cards, tokens or biometric means, should be used.

Authentication information should be accompanied by additional authentication factors for accessing critical information systems (also known as multi-factor authentication). Using a combination of multiple authentication factors, such as what you know, what you have and what you are, reduces the possibilities for unauthorized accesses. Multi-factor authentication can be combined with other techniques to require additional factors under specific circumstances, based on predefined rules and patterns, such as access from an unusual location, from an unusual device or at an unusual time.

Biometric authentication information should be invalidated if it is ever compromised. Biometric authentication can be unavailable depending on the conditions of use (e.g. moisture or aging). To prepare for these issues, biometric authentication should be accompanied with at least one alternative authentication technique.

The procedure for logging into a system or application should be designed to minimize the risk of unauthorized access. Log-on procedures and technologies should be implemented considering the following:

- a. not displaying sensitive system or application information until the log-on process has been successfully completed in order to avoid providing an unauthorized user with any unnecessary assistance;

Features are implemented within established UI deployments that have the appropriate authentication requirements for their feature-set. The customer-facing and internal management UIs require authentication before they will display anything other than the login page.

- b. displaying a general notice warning that the system or the application or the service should only be accessed by authorized users;

- c. not providing help messages during the log-on procedure that would aid an unauthorized user (e.g. if an error condition arises, the system should not indicate which part of the data is correct or incorrect);

- d. validating the log-on information only on completion of all input data;

- e. protecting against brute force log-on attempts on usernames and passwords [e.g. using completely automated public Turing test to tell computers and humans apart (CAPTCHA), requiring password reset after a predefined number of failed attempts or blocking the user after a maximum number of errors];

- f. logging unsuccessful and successful attempts;

QUESTION: Do we do this? Should we?

- g. raising a security event if a potential attempted or successful breach of log-on controls is detected (e.g. sending an alert to the user and the organization's system administrators when a certain number of wrong password attempts has been reached);

QUESTION: Do our third-party auth providers do this?

- h. displaying or sending the following information on a separate channel on completion of a successful log-on:

1. date and time of the previous successful log-on;

2. details of any unsuccessful log-on attempts since the last successful log-on;

QUESTION: I think we capture the time of the login, need to confirm. I don't think we have any visibility of unsuccessful logins as they are dealt with by the auth providers.

- i. not displaying a password in clear text when it is being entered; in some cases, it can be required to de-activate this functionality in order to facilitate user log-on (e.g. for accessibility reasons or to avoid blocking users because of repeated errors);

We override the default login pages in our auth providers, but we use the standard HTML "password" input field specifically for this.

- j. not transmitting passwords in clear text over a network to avoid being captured by a network "sniffer" program;

- k. terminating inactive sessions after a defined period of inactivity, especially in high risk locations such as public or external areas outside the organization's security management or on user endpoint devices;

- l. restricting connection duration times to provide additional security for high-risk applications and reduce the window of opportunity for unauthorized access.

## Other information

Additional information on entity authentication assurance can be found in ISO/IEC 29115.

## Control

Configurations, including security configurations, of hardware, software, services and networks should be established, documented, implemented, monitored and reviewed.

## Purpose

To ensure hardware, software, services and networks function correctly with required security settings, and configuration is not altered by unauthorized or incorrect changes.

## Guidance

### General

- The organization should define and implement processes and tools to enforce the defined configurations (including security configurations) for hardware, software, services (e.g. cloud services) and networks, for newly installed systems as well as for operational systems over their lifetime.
  - Roles, responsibilities and procedures should be in place to ensure satisfactory control of all configuration changes.

[Terraform](#) is used to implement infrastructure-as-code. This means that the configuration of our operational systems and cloud infrastructure is stored in our [code repository](#), and is under exactly the same [change controls](#) as all other code. Where there are exceptions, e.g. the services we use for [Authentication of End Users](#), these require additional documentation, that documentation being stored in the same code repository as all other code and configuration. We always favour services that can be fully configured via Terraform.

### Standard templates

Standard templates for the secure configuration of hardware, software, services and networks should be defined:

- using publicly available guidance (e.g. pre-defined templates from vendors and from independent security organizations);
- considering the level of protection needed in order to determine a sufficient level of security;
- supporting the organization's information security policy, topic-specific policies, standards and other security requirements;
- considering the feasibility and applicability of security configurations in the organization's context.

[Terraform](#) supports modularisation through parameterised resource definitions that can be reused to create multiple similar resources, e.g. for different regions. This means we are able to capture our current defined best practise for configuring a given type of resource and ensure it is consistently applied across our estate.

The templates should be reviewed periodically and updated when new threats or vulnerabilities need to be addressed, or when new software or hardware versions are introduced.

The templates are under constant review and enhancement by our dedicated Platform team, whose primary focus is on improving security controls.

The following should be considered for establishing standard templates for the secure configuration of hardware, software, services and networks:

- minimizing the number of identities with privileged or administrator level access rights;

Identities are defined as resources in infrastructure-as-code, and granted specific permissions as required for their purpose

- disabling unnecessary, unused or insecure identities;

Should a product refinement result in an identity becoming redundant it will be deleted from the Terraform configuration as part of that change, which will cause it to be automatically destroyed in the environment.

- disabling or restricting unnecessary functions and services;

Configuration as code makes this process automatic.

- restricting access to powerful utility programs and host parameter settings;

Access to most platform capabilities is controlled through [Azure DevOps](#) permissions against authenticated users. Exceptions (such as

- synchronizing clocks;

Implemented by NTP implementation built into [Azure](#) platform.

- changing vendor default authentication information such as default passwords immediately after installation and reviewing other important default security-related parameters;

PaaS resources require such settings to be explicitly provided during creation, so that they are secure by default. Resources are defined in Terraform modules wherever possible, which provides and maintains all their settings.

- invoking time-out facilities that automatically log off computing devices after a predetermined period of inactivity;

Our computing devices are managed by IT using [Azure Active Directory](#) as the identity provider.

- verifying that license requirements have been met (see 5.32).

PaaS resources are automatically billed through a payment system based on their licensing requirements.

## Managing configurations

Established configurations of hardware, software, services and networks should be recorded and a log should be maintained of all configuration changes. These records should be securely stored. This can be achieved in various ways, such as configuration databases or configuration templates.

This is implemented through [infrastructure-as-code](#), meaning that resources are defined in machine-readable files that are automatically applied to test and production environments. This means that the configuration of our operational systems and cloud infrastructure is stored in our [code repository](#), and is under exactly the same [change controls](#) as all other code. Where there are exceptions, e.g. the services we use for [Authentication of End Users](#), these require additional documentation, that documentation being stored in the same code repository as all other code and configuration. We always favour services that can be fully configured via Terraform.

Changes to configurations should follow the change management process (see [8.32](#)).

Configuration records can contain as relevant:

1. up-to-date owner or point of contact information for the asset;
2. date of the last change of configuration;
3. version of configuration template;
4. relation to configurations of other assets.

All of this is captured via infrastructure-as-code.

## Monitoring configurations

Configurations should be monitored with a comprehensive set of system management tools (e.g. maintenance utilities, remote support, enterprise management tools, backup and restore software) and should be reviewed on a regular basis to verify configuration settings, evaluate password strengths and assess activities performed.

Capabilities for maintenance and management, backup, failover etc. is supported by our PaaS providers, as is remote support (all resources are only ever accessed remotely, and in normal operation they are only managed by deployments via infrastructure-as-code).

Actual configurations can be compared with the defined target templates. Any deviations should be addressed, either by automatic enforcement of the defined target configuration or by manual analysis of the deviation followed by corrective actions.

Automatic enforcement is used in all cases where [infrastructure-as-code](#) is employed, though there are exceptions that have to be managed manually.

## Other information

Documentation for systems often records details about the configuration of both hardware and software.

System hardening is a typical part of configuration management.

Configuration management can be integrated with asset management processes and associated tooling.

Automation is usually more effective to manage security configuration (e.g. using infrastructure as code).

Configuration templates and targets can be confidential information and should be protected from unauthorized access accordingly.

## \WIP---NEW-SDLC\Infrastructure-and-Patterns.md

Documentation for what/how we use various products and resources to support the software development process.

Broken out into separate pages so we can reference them from our [annotations on controls](#).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#).

Our main provider of Platform as a Service (PaaS). Application components are deployed as containerised/virtualised packages that run inside fully managed facilities. All we need to specify is the memory and processing power requirements and the PaaS provider ensures our components execute inside a suitable instance.

Some services are even more virtualised, e.g. Azure SQL, in which the primitives are databases and elastic pools (they have "servers" but these are just logical collections of databases). The higher the level of abstraction, the fewer low-level details we have to be responsible for.

This eliminates the need to e.g. create dedicated machines that need to be patched or otherwise configured on a low level.

As a result of this abstraction, some of the details of procedural guidance refer to issues that, while they could affect us, we have outsourced the solution to Azure.

We use Microsoft's [PIM](#) to manage temporary elevation of permissions.

This allows us to normally restrict permissions for engineers to avoid accidental access, but to allow just-in-time privileged access where required in a way that is:

- time limited
- approved by peers
- audited with justification

## Base Permissions on Live

*Forthcoming...*

## Available Elevations

*Forthcoming...*

## Review Process

*Forthcoming...*

We use [Terraform](#) to implement the popular [Infrastructure as code](#) pattern.

This means that the exact configuration of (ideally) all the resources and services in our estate are automatically aligned with modular definitions held in our [code repository](#).

For this reason, we always favour PaaS providers for which Terraform integrations exist.

There are notable exceptions in the services we depend on:

- [Authentication of End Users](#)

As a result, the correct configuration for these services need to be documented in human-readable instructions in our code repository. This at least ensures that those instruction are under [change control](#).

[Azure Active Directory](#) is FISCAL's general solution to authenticating employees, and this is used to identify engineers (and any other users requiring access) in [Azure DevOps](#), and in any other services where single-sign-in integration is supported.

We use [Terraform](#) to create AAD groups that map to capabilities in our platform. The full list is defined in:

- [Infrastructure/modules/deploy-global/all.main.tf](#)

They follow a naming pattern:

```
nwg-<env>-<area>[-<role>]
```

- The <env> is the environment identifier, which is one of `live`, `try`, `perf` in our Azure deployments.
- The <area> describes some piece of the product architecture.
- The <role> describes the capabilities granted to members of the group, often `readers` or `writers`, and is optional as an area may not need to have multiple roles.

Any list reproduced here is liable to become out of date but an example set of groups, using `live` as an example of an environment, is as follows:

```
nwg-live-support
nwg-live-admins
nwg-live-ado-deployers
nwg-live-log-readers
nwg-live-log-writers
nwg-live-biblioteca-readers
nwg-live-biblioteca-writers
nwg-live-abed-readers
nwg-live-abed-writers
nwg-live-ml-workers
nwg-live-regionmanager-readers
nwg-live-regionmanager-writers
nwg-live-customersql-readers
nwg-live-customersql-writers
nwg-live-usagemonitoring-readers
nwg-live-usagemonitoring-writers
nwg-live-acr-reader
nwg-live-view-database-state"
```

Note that other groups exist with `live` replaced with `try` or `perf` throughout. This means that granting a user membership of a group can never simultaneously grant them the same kind of access in multiple environments; each environment is managed completely separately (although in a consistent way).

We use this to:

- host all source code in [git](#)
- record all work items, including [deployment authorisation](#)
- manage the creation, review and completion of [Pull Requests \(PRs\)](#)
- execute builds for PRs and deployments
- execute deployments
- manage permissions of individual users to do all the above

Users are authenticated by [Azure Active Directory](#).

The source code repository uses [git](#). This captures the full history of who has changed the source code and at what time.

We have one permanent shared branch:

- [Development branch](#)

We accumulate branches that document what was deployed to live:

- [Live branches](#)

Individual developers create short-lived (~1 day) feature branches for their [Pull Requests](#).

Our release pipelines automatically tag specific versions in the code when they are deployed to an environment, according to a naming convention:

<env>-<build>

where <env> is try, live, etc, and <build> is the build number generated by ADO.

The Development branch in our [git](#) repository contains our single "latest-and-greatest" code base. Thanks to NXG being a purely cloud-deployed product we do not have to maintain long term branches for old versions.

Our [Pull Request](#) process is the only means for updating the Development branch. [Azure DevOps](#) configuration for that branch ban users from pushing commits directly to it.

Live branches are short term (~2 weeks) maintenance branches off the [Development branch](#). One is created at the end of a sprint when we are preparing a [Planned Release](#).

They are named `live/<sprint>` where `<sprint>` is the number of the sprint that has just been completed (i.e. the sprint that produced what is being released).

The build that is deployed to live can either be built from the just-created `live/<sprint>` branch, or else we can deploy the latest Development build to live and later create the `live/<sprint>` branch by referring to the commit ID.

During the following sprint we continue to merge changes to the Development branch, in preparation for the next planned release, accumulating changes.

It is often the case that before the next planned release, we need to do an [Ad hoc Release](#). This is the purpose of the live branch: it serves as a place to [cherry-pick](#) specific code changes into, rather than us being forced to release all the changes in the Development branch early. After cherry-picking the required changes into the live branch, we can build and deploy to live.

This process has to be authorised through a [Release work item](#).

When the branch `live/N+1` has been deployed, the branch for `live/N` is no longer updated and is retained in case it might be useful for future reference.

[Pull requests](#) are a standard mechanism in ADO to support the review process for code changes.

Before a set of code changes can be committed to our [Development](#) branch it must go through the pull request review process:

- It MUST be linked to a work item
- It MUST pass unit and DB integration tests (this implies a successful automated build)
- It MUST be approved by at least one reviewer (see guidance below)

These requirements are automatically enforced by ADO.

## Workflow

This is the typical "feature branch" pattern:

- Engineer creates a branch off of [Development](#). They MAY prefix the name with a work item number, to make it easy to ensure the name is unique.
- Engineer makes commits to their branch, which is known as a feature branch. They can test their changes in their [local environment](#).
- When they are ready to merge, they create a PR for their branch to be merged into [Development](#). The PR name SHOULD begin with a work item number in the format #12345.
- When the automatically enforced requirements are met, the merge will be allowed. This will usually happen automatically if auto-complete is enabled on the PR.
- The feature branch can then be deleted. This will usually happen automatically.

The Engineer should check for when their change has been auto-deployed to the `try` environment, and carry out basic checks to ensure that it is behaving as expected.

## Guidance for Reviewers

- things to look out for
- especially security risks

- details of on-premise hardware environment?
- some are cloud deployed
- all use standard Microsoft image

- PR builds
- CI builds
- CI deployments (incl. UI smoke tests)
-

This is a specialised type of [ADO](#) work item that is used to manage the process of authorising a deployment to our production platform.

Two broad kinds of release are dealt with via the same work item template:

- [Planned Releases](#) are aligned with our sprint cycle.
- [Ad hoc Releases](#) occur whenever required based on customer need, bugs etc. at a point when we may not be ready to release all changes made since the previous planned release.

(Note: deployments to [Test Environments](#) are fully automated and do not require manual authorisation).

## Approvers

An important manual convention is that individuals who have approved the release MUST add their own name to the *Approvers* section, i.e. users MUST NOT add anyone else's name to that section.

This means that the change history for the work item will show the action of approving as being taken by that user.

Planned releases are biweekly, usually on the Monday as a new sprint begins, to deploy all of the prior sprint's work to the [Production Environment](#).

The [Release work item](#) authorising the release is discussed in the [Release Readiness Meeting](#) on the previous Thursday so we know well in advance if there are any issues affecting the normal release schedule.

The [live branch](#) for the release is soon after the readiness meeting so we have a static picture of what will be released and that it will be identical to what is deployed to [try](#).

If no subsequent last-minute changes are required, we can deploy using the same build previously deployed to [try](#). Otherwise we can do a new build from the live branch.

Once a successful build has been selected, we can run the [deploy pipeline](#) to target the [Production Environment](#).

An ad hoc release is often required to address a support issue, or make a small, localised enhancement for some urgent customer requirement.

There are two broad classes of ad hoc release:

- Terraform-only
- Code changes

## Terraform-only

*Details?*

### Code changes

During the sprint following a [Planned Release](#), we continue to merge changes to the [Development branch](#) branch, in preparation for the next planned release, accumulating changes.

During this period, we do not necessarily want to be forced to release all those changes yet (this gives us some freedom to tackle larger pieces of work in stages.) So for an ad hoc release involving code changes, we [cherry-pick](#) specific code changes into to the previous sprint's [Live branch](#).

This process has to be authorised through a [Release work item](#).

We reference third-party packages in public registries such as [npm](#), [Nuget](#) and [PyPI](#).

- process for maintaining/updating?

We integrate with two external systems to take care of authenticating our product's end users:

- [Azure Active Directory B2C](#)
- [Auth0](#)

They are called out specifically here because they are examples of infrastructure that we cannot manage purely with [Terraform](#). We have to document how they are configured.

This is called out as an exceptional resource because it is not linked into our organisation's [Azure Active Directory](#), so permissions have to be controlled separately by IT.

Our test environments include:

- [Try](#) is our continuous integration testbed
- [Perf](#) is used for weekly automated performance tests

Try, accessible via <https://cloud-euwest.tryapfnxg.com/>, is our continuous integration testbed.

## Open Permissions

Permissions on this environment are as open as possible so it is continuously accessible to all engineers, to make it possible for engineers to troubleshoot new features and observe how they interact with the evolving background product.

So for example, any database may be examined to check that a feature is storing the expected data, or data may be modified to set up an exploratory test scenario, simulate a fault that must be tolerated, etc.

Similarly all Azure resources are accessible so engineers can be educated about the operation of any aspect of the platform.

## No Sensitive or Customer Data

To meet its purpose as an open environment for learning about and enhancing the operation of our product(s), it is obviously vitally important that no real data obtained from customers of any kind may be used on this environment.

All data must be "dummy". We have tools that can bulk-generate such data randomly, and these tools should be enhanced and extended to cover new requirements for data formats and common test cases.

Perf, accessible via <https://cloud-euwest.perfapfnxg.com/> is used for weekly automated performance tests.

## Closed Permissions

This environment is not available for engineers to explore. For general experimentation and education purposes, the [try environment](#) must be used.

Our security stance for this platform is identical to the live platform, as this allows us to use it to simulate processing runs with snapshots of actual customer data.

## Automated Test Runs

To remove the need for engineers to access the platform or the test data directly, the [Performance-Test](#) pipeline automatically:

- creates a test customer,
- uploads the test data from the [perfdatanxg](#) storage account (in the [NXG](#) subscription)
- waits for processing to complete
- deletes the test customer

The overall timing for a run can be read from the pipeline's results. More detailed timing information for specific Hangfire jobs appears in the environment's own [App Insights logging](#).

## Acquiring New Test Data

Updated test data must be acquired from a real customer. This must be authorised by a [Zendesk](#) ticket.

Local environments use Docker to emulate our platform dependencies, allowing every aspect of the product to be developed, tested, fixed and enhanced in isolation from any other engineer using only resources running on the engineer's personal machine.

There are some external services in the [try\\_environment](#) that are depended on to support local environments, but which can be used without requiring any special configuration to support each local environment, so there is no risk of "cross-contamination":

- [Authentication of End Users](#)
- Azure Vision (OCR service used by PDF/image statement matching)

These dependencies are automatically set up and work mostly transparently.

Refer to our [Local Environment development guide](#) for usage instructions.

The production environment uses the name `live`, and is accessible via the regional Cloud UI pages:

- <https://cloud-euwest.apfnxg.com>
- <https://cloud-uksouth.apfnxg.com>
- <https://cloud-uswest.apfnxg.com>

We use Zendesk as a catch-all mechanism for recording any decisions or authorizations that don't take place in a more suitable/specific framework.

## **WIP---NEW-SDLC\Infrastructure-and-Patterns\Release-Readiness-Meeting.md**

This is a biweekly meeting on the second Thursday of each sprint, so at least one working day before the next full deployment.

It gives senior engineers and IT a forum to go through release readiness checks and agree that the work of the sprint is ready to deploy to the [Production Environment](#) as a [planned release](#).

