ADFR001 Test Strategy

The aim of this document is to describe a simple and robust Test Strategy for project *<Project Name>*. It includes or references the guidance and standards that will inform:

1. Code reviews
2. Software design decisions at all levels
3. The technical stack used for Test Frameworks
4. The level of quality assurance within the software delivery lifecycle (SDLC)

Version Control

This is a living document that will evolve over the lifecycle of any project. It must be contained within the project repository and version controlled accordingly (using Git). The first draft of this document must be agreed before any project commences.

Ownership

This document will be owned and maintained by Development, with assistance and independent oversight from the Test Team.

High Level Test Strategy

The table below describes the types of testing that must be performed during the Agile SDLC:

Test Types

Name	Manual or Automated	Description
Unit Testing	Automated	Tests for individual units of code. Any interfaces or dependences are mocked.
Functional Testing	Automated	Tests to ensure that the logic coded into the application fulfils the user requirements. These are currently called 'Integration Tests' or 'Deep Tests'.
E2E Testing	Automated	Tests to exercise the entire application stack. Typically run from the application UI and mimic user interaction.
Visual Testing	Automated	Tests to ensure that the visual elements of the application are correct. This is not tested by E2E or functional tests.
Exploratory Testing	Manual	Tests to ensure that unexpected user activity does not result in unhandled application failure.

Each of these Test Types has a dedicated section within this document below.

The table below describes how these types of testing integrate with a typical Agile SDLC:

Test Mapping to SDLC

SDLC Phase	Test Process
Sprint Planning	BDD Scenarios created
During the Sprint	Unit Tests, Functional Tests and E2E Tests completed (Automated).
Outside of the Sprint	Exploratory Testing performed (Manual)

Continuous Integration

Describe here when the different automated regression tests are run in the CI pipeline. E.g.

Continuous Integration

Action	Regression Testing Performed

Action	Regression Testing Performed
Push to Branch	All Functional Tests and Visual Tests are run for that Feature.
Merge to Release Branch	All Functional Tests run for all Features, all E2E tests run, all Visual Tests are run.

Defect Management

With the exception of Exploratory Testing, all issues with the code identified by failing tests will be corrected within the Sprint and not documented as defects.

Unit Testing

Aim

Unit Tests are the only way code can be monitored against breaking changes (mutations) in the code.

However, they can also be used to document the design of the code, eliminating the need to capture and maintain this information elsewhere.

The guidance below covers both uses.

Guidance

Code Design

The application will be built of standard components that perform different functions (for example, the MVC (Model, View, Controller) design model). The design should be captured here at a high level.

Templated or example Unit Tests should be linked to within this document and describe each of the typical components of the software design. They should

describe the mocks, stubs and spies that can be used with them. Example assertions on the logic, structure and data should be included.

If this is not possible, or exceptions are needed, this should be captured here, along with a strategy to standardise the design in the future.

These templates should be used to enforce good code design, inform code reviews and enable TDD (Test-Driven Development) if the developer uses this methodology.

Test Framework

A description of the Technical Stack of the Unit Test framework and a brief description of how it is implemented must be included here to support the developers.

Tech Stack

Tool	Description	Use
Karma	Test Runner	
TBC		

Coverage and Reporting

Test coverage is currently measured through Karma and can be used to inform Code Reviews to ensure that a developer is writing unit tests.

Continuous Improvement

Issues and Improvements

Improvement	Detail	Owner	Status

Improvement	Detail	Owner	Status
Test Coverage to be included in CI	Test coverage will be included in SonarQube and CI. However it is emphasised that this must not be used as an indicator of code quality in isolation. Ideally this will be used to inform Development of their own quality processes, and not reported throughout the business as it can be easily mis-interpreted.		
Mutation Testing	Mutation testing needs to be included in the CI pipeline. A PoC needs to be created to identify how best this can be used where best to include it in the Agile SDLC. Again, this will inform the Development team of the effectiveness of their quality processes, and is usually only shared with stakeholders who can correctly interpret the results.		

BDD (Behaviour-Driven Development)

Aim

Behaviour-Driven Development ensures that the system is designed around the functional requirements of the user. It also ensures functional issues are identified as early as possible, typically when the business logic is created.

Guidance

The following simple process will be followed for any User Story.

Please note: the BDD process below doesn't include the CI or Code Control quality processes (e.g. code reviews).

BDD Process

Step	Step #	Owner	Output
A Feature File is created containing the User Story	BDD01	Product Owner	Feature File containing the User Story.
An Example Mapping session is held	BDD02	Product Owner	Acceptance Criteria added to Feature File
Scenarios are created	BDD03	Developer and Tester	Scenarios added to Feature file
Scenarios are reviewed by Product Owner	BDD04	Product Owner	Updated scenarios (if necessary)
Technical implementation discussion	BDD05	Developers	Technical implementation notes are captured and E2E scenarios are identified.
Scenarios are implemented in the code	BDD06	Developers	Functional Tests covering all scenarios are created and test report is green. Unit Tests have been created and test report is green.
Any E2E test scenarios are implemented	BDD07	Developer / Tester	E2E tests. E2E test report is green.

Functional Testing

Functional Testing will be performed to ensure that all user requirements have been met by the functionality of the system.

A description of the Technical Stack of the Functional Test framework and a brief description as to how it is implemented must be included here to support the developers.

Tech Stack for Functional Testing

Tool	Description	Use

Tool	Description	Use
Karma	Test Runner	
TBC		

E2E Tests

End to end tests will be identified during the Technical Review (step BDD05 of the BDD process) by the following criteria:

1. The middleware API is triggered
2. The scenario covers business functionality that is provided by Guidewire or the middleware.
3. The scenario is High Risk or High Impact

A description of the Technical Stack of the E2E Test framework must be included here to support the developers and Testers.

Tech Stack for E2E Tests

Name	Tags	Files
Protractor		
TBC		

Continuous Improvement

There is currently no cross-browser testing performed. It may be more effective to do this with any Visual Testing framework that is created (see next section). If cross-browser testing is performed with the E2E framework, then it may be useful to mock any middle tier and network responses, to maximise the speed and reliability of these tests.

There is currently no integration into the CI environment or any 'tagging' strategy for scenarios.

Visual Testing

Aim

The design of the GUI is provided by a third party. This includes the UX design and accessibility. Therefore this design must match what is eventually implemented.

In addition, any unnecessary changes to the DOM will disrupt E2E test automation and may also affect UX, accessibility or the requirements.

Guidance

No visual testing is currently being performed.

Continuous Improvement

Snapshot testing of the DOM with Karma can be used to ensure it doesn't change accidentally. Visual Testing could be performed with BackstopJS, Gemini or Happo (and/or Applitools).

Exploratory Testing

Aim

Exploratory Testing will capture any application failures due to unusual user behaviour.

Guidance

A simple procedure for exploratory testing must be captured here. It must describe the methodologies used (e.g. boundary testing), links to guidance documents for the application and the tools needed (e.g. chrome plugins for recording your test session in a video).

Defect Management

The defect management process must be included here also. It must describe how defects are captured, reviewed, assigned and included in a sprint.