



**Document Purpose**:

This document captures and communicates high-level requirements, initial solution design, and design considerations during initial analysis.  This allows the project team to properly scope the integration.  This document is also used as an input into the accompanying Detailed Design document produced during the project's development phase.

**Document History**:

| **Version #** | **Date** | **Author** | **Summary of Change** |
| --- | --- | --- | --- |
| 1 | 01/30/2024 | S Kolla | Initial Draft |
|  |  |  |  |

**Contents**

[Section 1: Introduction and Requirements 5](#_heading=h.30j0zll)

[Introduction 5](#_heading=h.3znysh7)

[Overview 5](#_heading=h.2et92p0)

[Purpose 5](#_heading=)

[Scope 5](#_heading=h.5ybigxav2zh5)

[Audience 5](#_heading=h.3dy6vkm)

[Terminology 6](#_heading=)

[Business Requirements 6](#_heading=h.4d34og8)

[User Stories 6](#_heading=h.17dp8vu)

[Section 2: High Level Design 6](#_heading=h.26in1rg)

[Summary: 6](#_heading=h.lnxbz9)

[Section Purpose: 7](#_heading=h.1ksv4uv)

[Business Process Diagram 7](#_heading=h.z337ya)

[Functional Overview 7](#_heading=h.3j2qqm3)

[Solution Overview 7](#_heading=)

[Context Diagram 7](#_heading=)

[Architecture 7](#_heading=h.pn8ndxx2x0kg)

[Technology Stack 8](#_heading=h.ddkrqsep07n2)

[High Level Component Description 8](#_heading=)

[Security 8](#_heading=h.2bn6wsx)

[Technical Debt 9](#_heading=h.qsh70q)

[Assumptions, Constraints and Risks 9](#_heading=h.3as4poj)

[Business Assumptions 9](#_heading=)

[Technical Assumptions 9](#_heading=)

[Dependencies 9](#_heading=)

[Section 3: Detail Level Design 9](#_heading=h.1pxezwc)

[Solution Overview 9](#_heading=h.2p2csry)

[Component Description 9](#_heading=h.147n2zr)

[Design Diagrams 10](#_heading=h.32hioqz)

[Class Diagram 10](#_heading=)

[Sequence Diagram 12](#_heading=h.ihv636)

[Activity Diagrams 13](#_heading=)

[Database Configuration 13](#_heading=h.41mghml)

[RentalItemStatus Enumeration 14](#_heading=h.2grqrue)

[Contains all the valid status for a rental item 14](#_heading=h.ey02wmqdpp8w)

[External Libraries 14](#_heading=h.1v1yuxt)

[3rd Party Jars 14](#_heading=h.2nusc19)

[Data Validation 14](#_heading=h.tfnq80bn023u)

[Controller Method Parameters: 14](#_heading=h.3koanfywwgl2)

[DTO (Data Transfer Object) Validation: 14](#_heading=h.b2fyd56xbz7d)

[Service Layer: 14](#_heading=h.dzy0i9fcrzr5)

[UI Configuration 15](#_heading=h.19c6y18)

[Rental Web Service 15](#_heading=h.28h4qwu)

[displayTools Operation 15](#_heading=h.37m2jsg)

[rentTool Operation 15](#_heading=h.e9o23g6zy5km)

[returnTool Operation 15](#_heading=h.4688zyx5wtoy)

[extendRental Operation 15](#_heading=h.gi1idfey4qhf)

[cancelRental Operation 16](#_heading=h.ps1nv1jxljmb)

[Batch Process 16](#_heading=h.3ygebqi)

[Initial Conditions 16](#_heading=h.sqyw64)

[Action 16](#_heading=h.3cqmetx)

[Batch Process Details 16](#_heading=h.1rvwp1q)

[Report Specification 16](#_heading=)

[Design Considerations 16](#_heading=h.48pi1tg)

[Logging 16](#_heading=h.nwemlbk3p9ya)

[Known Issues 16](#_heading=h.1302m92)

[Planned Future Enhancements 16](#_heading=h.3mzq4wv)

[Performance and Availability 16](#_heading=h.2250f4o)

[Data Warehouse Impacts 17](#_heading=h.haapch)

[Supplementary Specifications Impacted 17](#_heading=h.319y80a)

[Conversion and Production Impacts 17](#_heading=h.1gf8i83)

[Concurrency Considerations 17](#_heading=h.40ew0vw)

[Clustering Considerations 17](#_heading=h.2fk6b3p)

[Operating System Considerations 17](#_heading=h.upglbi)

[Security 17](#_heading=h.3ep43zb)

[Technical Debt 17](#_heading=h.1tuee74)

[External Systems 17](#_heading=h.4du1wux)

[Additional Considerations 17](#_heading=h.2szc72q)

[Errors and Resolution 17](#_heading=h.184mhaj)

[Testing the Solution 17](#_heading=h.3s49zyc)

[Unit Test Classes 17](#_heading=h.cixw13ue1xzy)

[Unit Test Class 1 17](#_heading=h.lupdky9ffxk9)

[Conclusion 17](#_heading=h.ah5svou0cds1)

[Section 4: Resources and Review 18](#_heading=h.279ka65)

[Additional Resources 18](#_heading=h.meukdy)

[Appendix 18](#_heading=h.36ei31r)

[Notes 18](#_heading=h.1ljsd9k)

[Elaboration Questions 18](#_heading=h.45jfvxd)

[Assumptions, Constraints and Risks 18](#_heading=h.2koq656)

[Business Assumptions 18](#_heading=)

[Technical Assumptions 18](#_heading=)

[Review and Acceptance 18](#_heading=)

[Roles and Responsibilities for reviewers of this vision Document 18](#_heading=h.1yyy98l)

[Approvals 19](#_heading=h.2y3w247)

[Sign Off 19](#_heading=h.1d96cc0)

# Section 1: Introduction and Requirements

## Introduction

### Overview

The Tool Rental System is a comprehensive software solution designed to manage and automate the processes involved in renting tools from a tool rental business. This document outlines the architecture of the system, defining its components, their interactions, and the overall structure of the application while emphasizing the exclusion of holidays from rental calculations and business processes.

### 

### Purpose

The primary purpose of the Tool Rental System is to streamline the tool rental process, from inventory management to customer interactions, providing an efficient and user-friendly platform for both customers and the tool rental business. In addition the Tool Rental System excludes holidays from rental duration calculations and scheduling. This enhancement ensures accurate rental durations and prevents tool pickups or returns on holidays.

### Scope

The system encompasses functionalities such as:

* Inventory Management: Tracking and managing tools, their availability, and condition.
* Customer Management: Registration, authentication, and history tracking for customers.
* Holiday Management: Ability to define and manage holidays within the system.
* Rental Management: Handling tool rental requests, reservations, and returns.
* Exclusion of Holidays: Rental durations and availability calculations exclude holidays.
* Reporting and Analytics: Providing insights into rental patterns, popular tools, and revenue generation.

This design document focuses only Rental Management, for details related to the other functionalities, please refer to the associated design documents

### Audience

This document is intended for technical integration developers and consultants who will be implementing this integration. It is also intended for implementation managers, functional integration consultants, business analysts, subject matter experts, quality assurance analysts, and client management. The reader should be familiar with the tools and techniques used to drive this design, as well as Gosu, Java, XML, and Web Services.

### Terminology

| **Term** | **Definition** |
| --- | --- |
| REST | Representational State Transfer, an architectural style making it easier for client and server computer systems to communicate with each other |
| Inventory Management System | A system where inventory of tools are maintained, internal to the organization |
| Holiday Calendar | A system where holidays are maintained, internal to the organization |

## Business Requirements

### User Stories

User stories can be kept at a higher level or brought in from the project’s user story repository. Include specific details as needed, along with associated acceptance criteria. For out-of-scope user stories, only capture those stories that were key scope decisions.

| **User Story ID** | **Name** | **Description** | **Acceptance Criteria** |
| --- | --- | --- | --- |
| US01 | List tools | As an end user, I should be able to see a list of tools available for rent | List of tools are displayed |
| US02 | Rent a tool | As an end user, I should be able to rent a tool for desired number of days | Tool is rented |
| US03 | Return a tool | As an end user, I should be able to return the tool | Tool is returned |
| US04 | Extend rental | As an end user, I should be able to extend the rental for desired number of days | Rental is extended |
| US05 | Cancel rental | As an end user, I should be able to cancel the rental after initial request | Rental is canceled |

# Section 2: High Level Design

## Summary:

Rental Management System facilitates tool rental.  This design document provides a foundational framework for integrating with Inventory System and Holiday Calendar to allow for the checkout processes.

## Section Purpose:

This section's purpose is to capture and communicate high-level requirements, initial solution design, and design considerations during inception.  This allows the project team to properly scope the integration.  This section is also used as an input into the Detailed Design section.

## Business Process Diagram

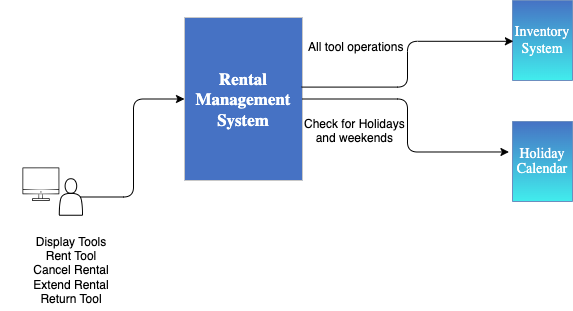
NA

## Functional Overview

Please refer to [User Stories](#_heading=h.17dp8vu) Section

## Solution Overview

### Context Diagram



### 

### Architecture

The Tool Rental System adopts a three-tier architecture:

* Presentation Tier: This tier handles user interaction and interfaces with customers and staff. It includes the web-based user interface and mobile applications for ease of access.
* Application Tier: This tier consists of the application server responsible for processing business logic, managing transactions, and coordinating communication between the presentation and data tiers.
* Data Tier: The data tier manages the storage and retrieval of data. It includes the database server where tool, customer, and transaction data are stored.

### 

### Technology Stack

* Frontend: <list the front end technologies here JSP or React etc.>
* Backend: Java with Spring Framework for REST APIs, and an application server.
* Database: MySQL (or equivalent) for relational data storage.
* Authentication: Basic authentication and / or JSON Web Tokens (JWT) for secure user authentication.
* Communication Protocol: RESTful API for data exchange between frontend and backend.
* Secure Socket Layer (SSL): Ensures secure communication over the web interface and mobile applications.

### High Level Component Description

The table below provides an overview of the components involved in the solution, including the associated user story(s). Each component is described in more detail in the sections below.

| **User Story** | **Component** | **Description** |
| --- | --- | --- |
| List tools | RentalManager  InventoryManager | RentalManager talks to InventoryManager and displays only the available tools |
| Rent a tool | RentalManager  InventoryManager  Holiday Calendar | RentalManager works with both inventoryManager and HolidayCalendar to perform calculations and create a rental. |
| Return a tool | RentalManager  InventoryManager | RentalManager returns the item and notifies InventoryManager |
| Extend rental | RentalManager  InventoryManager  Holiday Calendar | RentalManager works with both inventoryManager and HolidayCalendar to perform calculations and extend the rental. |
| Cancel Rental | RentalManager  InventoryManager | RentalManager cancels the future rental and notifies InventoryManager |

### Security

* Data Encryption: Use SSL to encrypt data during transmission.
* Authentication: Implement secure authentication mechanisms to protect user accounts and sensitive data.
* Authorization: Enforce proper access controls to ensure that users only access information relevant to their roles.

### Technical Debt

NA

### Assumptions, Constraints and Risks

#### Business Assumptions

1. Each rental contains only one tool, in the future, multiple tools might be included.

#### Technical Assumptions

1. Relevant data such as tools, customers and users are already loaded and available. Making this data available is outside the scope of this document.
2. Inventory system is capable of verifying the availability of tools based on the dates and also capable of updating the availability based on the rental. Implementing this logic is outside the scope of this design document.

#### Dependencies

| **Dependency** | **Impact of Missed Dependency** |
| --- | --- |
| Loading the appropriate data in external system | Rental system will not be able to access the information and hence can not perform rental transactions. |

# Section 3: Detail Level Design

* Following can be included in a Detail Level Design section of a combined document or break this document to high level and detailed level documents
* Audience: Architects and Delivery Team
* Purpose: To communicate the technical solution in a way that is executable by the delivery team with enough specificity to be used as both long term solution documentation and deliverable by a knowledgeable technician outside of the user who authored the document.

## Solution Overview

### 

### Component Description

The table below provides an overview of the components involved in the solution, including the associated user story(s). Each component is described in more detail in the sections below.

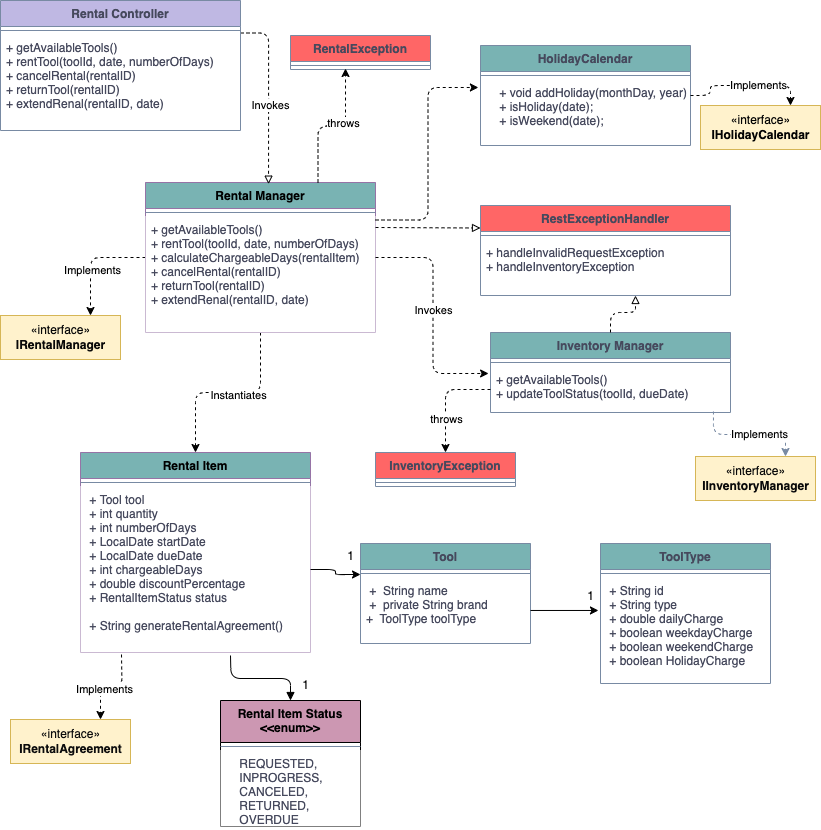
| **User Story** | **Component** | **Description** |
| --- | --- | --- |
| List tools | Rental REST API  RentalManager  InventoryManager | RentalController asks RentalManager which inturn gets the available tools. |
| Rent a tool | Rental REST API  RentalManager  InventoryManager  HolidayCalendar | RentalController passes the required information to RentalManager  RentalManager checks with InventoryManager for the availability of the tool, throws an exception if not available.  If available calculates the totals and discounts per the requirements  Updates the due date on the rental  Returns the display information |
| Return a tool | Rental REST API  RentalManager  InventoryManager | RentalController passes the required information to RentalManager  RentalManager recalculates the charges and updates necessary information on the rental.  RentalManager invokes InventoryManager so the inventory is updated and available for next rental. |
| Extend rental | Rental REST API  RentalManager  InventoryManager  HolidayCalendar | Same as Rent a tool |
| Cancel rental | Rental REST API  RentalManager  InventoryManager | RentalController passes the required information to RentalManager  RentalManager invokes InventoryManager so the inventory is updated and available for next rental. |

### 

### Design Diagrams

#### Class Diagram

The class diagram illustrates the basic structure of a tool rental system with its main components and their relationships. Please note that this shows simplified version of Inventory and Holiday management systems, and you may need to refer to the specific design documents for more details on those classes



#### 

| **Class** | **Description** |
| --- | --- |
| RentalManager | This class represents the tool rental system. It provides methods to list tools and for all the rental operations |
| InventoryManager | Manages tool availability, conditions, and inventory |
| HolidayCalendar | This class facilitates holiday management. It allows users to add and remove holidays |
| Tool | This class represents a tool. It has properties such as id, name, description, and rental rate. The class provides getter methods to access these properties |
| RentalItem | This class represents a tool rental. It has properties such as id, tool, customer, rental date, and return date. The class provides getter methods to access these properties. |
| RentalController | REST API which the front end application invokes |

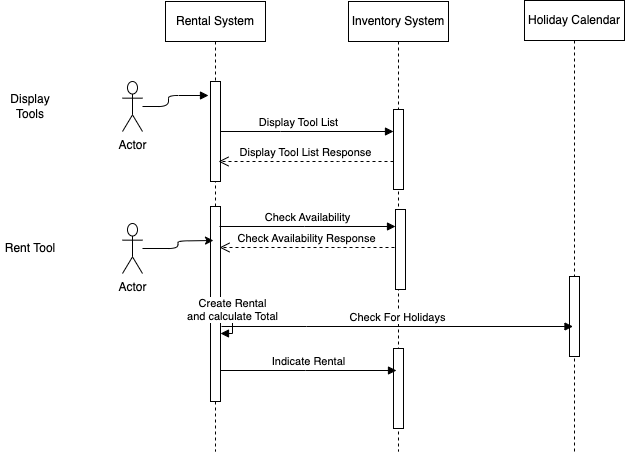
Controller Layer: All controller classes are under this category. These are the first line of defense for incoming data. You can use annotations directly on your model class (DTO) attributes to enforce constraints, such as `@NotNull`, `@Size`, `@Min`, `@Max`, etc. If these constraints are violated, a `MethodArgumentNotValidException` is thrown, which you can handle to send an appropriate response to the client. In cases where you want the validation to be outside of the controller then fields can be manually checked in the method implementations.

Service Layer: All the “manager” classes come under this category. More complex validations that require business logic or interaction with the database should be done in the service layer. This could be checking if a user with a certain email already exists, or if a product is in stock. If these validations fail, you can throw a custom exception, which can be caught and handled in the controller layer.

#### Sequence Diagram

Following sequence diagram shows the sequence of operations for each api operation. This depicts only the “Rent Tool” operation

<add other operations flows to the same sequence diagram>



#### Activity Diagrams

NA

#### Database Configuration

The system utilizes an RDBMS to store data. The database design includes the following tables:

**RentalItem Table**

- Stores rental transaction details.

- Fields: rental\_id, tool\_id, customer\_id, rental\_date, return\_date, rental\_fee.

There will be other tables related to holidays and inventory which will be utilized for the rental process, the design and description of those are found in the relevant design documents.

#### RentalItemStatus Enumeration

#### Contains all the valid status for a rental item

| **Code** | **Description** | |
| --- | --- | --- |
| REQUESTED | When initially requested for rental, item has this status | |
| INPROGRESS | After checkout is complete, status changes to this | |
| RETURNED | When return is complete | |
| OVERDUE | Overdue (A batch or a db script should update this status after due date is reached and rental is not returned. | |

#### External Libraries

<List all key/major external dependent libraries as part of this design>

#### 3rd Party Jars

<This section must list any 3rd party jars (open source, customer developed) used by the system and where the jars are deployed in the code. Alternatively, you can list the gradle or maven dependents>

* org.springframework.boot:spring-boot-starter-web (used for implementing REST)
* org.jetbrains:annotations:23.0.0 (used for validation)

#### Data Validation

There are different places where you can validate data types, and the choice often depends on the specific requirements of your application. Here are some common places to perform data type validation in a Java REST API:

##### Controller Method Parameters:

Validate data types as method parameters in your controller methods. This is often the first line of defense to ensure that the data received by the API is of the expected type.

Use annotations such as @RequestParam, @PathVariable, or @RequestBody with appropriate validation annotations like @Valid or @NotBlank from libraries like Hibernate Validator.

##### DTO (Data Transfer Object) Validation:

If your API uses DTOs to transfer data between the client and the server, you can perform validation on the DTOs themselves.

Use annotations from validation libraries on the DTO fields to enforce data type constraints.

##### 

##### Service Layer:

Perform additional validation in the service layer if there are business rules or complex validation logic that goes beyond simple data type checking.

Ensure that data passed to the service layer is valid before processing it.

#### UI Configuration

<Describe any UI configuration from a technical perspective. Any detail included here should correspond to the functional overview of the user stories>

#### Rental Web Service

A Spring framework controller class (RentController) which represents REST API, contains all the operations needed for rental transactions. Following operations are available on the controller class.

##### 

##### displayTools Operation

Invokes RentalManager which gets all the available tools from InventoryManager.

##### rentTool Operation

* Invokes RentalManager which checks with InventoryManager.
* InventoryManager encapsulates the logic for verifying the availability of the tool based on the ID, date and number of days etc.
* If not available an appropriate exception is thrown and the frontend will display the details accordingly.
* Validation of start date and number of days is done in the RentalManager class, InventoryManager is expected to handle the tool Id validation and the appropriate exception.
* Once the verification is complete, rental is created and summary is returned.

##### returnTool Operation

* Invokes RentalManager which verifies if rental exists.
* If it exists, notifies the InventoryManager.
* If it doesn’t exist, then an appropriate exception is thrown.
* InventoryManager encapsulates the logic for marking the tool is available.
* RentalManager marks the rental as returned or complete.

##### extendRental Operation

* Invokes RentalManager which verifies if rental exists.
* If it exists, notifies the InventoryManager of extension.
* If it doesn’t exist, then an appropriate exception is thrown.
* InventoryManager encapsulates the logic for verifying the availability of the tool based on the ID, date and number of days etc. and returns response.
* If not available an appropriate exception is thrown and the frontend will display the details accordingly.
* Validation of start date and number of days is done in the RentalManager class, InventoryManager is expected to handle the tool Id validation and the appropriate exception.
* Once the verification is complete, rental is created and summary is returned.

##### cancelRental Operation

* Invokes RentalManager which verifies if rental exists.
* If it exists, notifies the InventoryManager.
* If it doesn’t exist, then an appropriate exception is thrown.
* InventoryManager encapsulates the logic for marking the tool is available.
* RentalManager marks the rental as canceled (status changed to canceled).

#### Batch Process

<Describe this custom batch process. Repeat this section if this solution involves multiple custom batch processes>

##### Initial Conditions

<Describe the initial conditions, if any, for this batch process>

##### Action

<Describe the work performed by this batch process. Also describe any exceptions this batch process may encounter, and how these exceptions will be handled. Note the need for any special indexes to support queries that this batch process performs>

##### Batch Process Details

| **Batch Process Attribute** | **Value** |
| --- | --- |
| Batch Process Schedule |  |
| Batch Process Dependencies |  |
| Execution Deadline (if known) |  |
| Expected Run Time |  |

#### Report Specification

#### Design Considerations

#### Logging

All the rental classes will use the logger with category “Rental”. At the beginning and ending of the method do info level logging. Any information that can be used to debug an issue should be at the debug level.

### Known Issues

### Planned Future Enhancements

### Performance and Availability

| **Performance/Availability Requirement** | **Expectation** |
| --- | --- |
| Approximate time to communicate for each operation | 2 seconds |
| Web services | 24X7 except maintenance window |

### Data Warehouse Impacts

### Supplementary Specifications Impacted

### Conversion and Production Impacts

### Concurrency Considerations

### Clustering Considerations

### Operating System Considerations

### Security

<What security implications does this design include>

### Technical Debt

<How will this design impact technical debt>

### External Systems

### Additional Considerations

### Errors and Resolution

### Testing the Solution

## Unit Test Classes

### 

### Unit Test Class 1

<List all the operations and details of this class>

## Conclusion

The Tool Rental System's architectural design focuses on scalability, security, and a user-friendly experience. This document serves as a blueprint for developers, ensuring a systematic and well-organized implementation of the system.

# Section 4: Resources and Review

## Additional Resources

* Reference when appropriate to Charter
* Reference when appropriate to any auxiliary documents
* Reference when appropriate to any other design documents

## Appendix

### Notes

List additional notes here

### Elaboration Questions

| **Question** | **Date** | **Response** | **Date** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Assumptions, Constraints and Risks

#### Business Assumptions

1. <Add detailed level business assumptions here>

#### Technical Assumptions

1. <Add detailed level technical assumptions here>

## Review and Acceptance

### 

### Roles and Responsibilities for reviewers of this vision Document

| **Term** | **Definition** |
| --- | --- |
| **Peer Review** | Team to review process to ensure that correct technical or process steps are defined within current work practices. |
| **Oversight Review** | Team to review the process to ensure that regulatory, architecture or IS best practices or pre-defined guardrails are followed |
| **Stakeholder Review** | Team to review the process to ensure that we are meeting the current and business future business needs. |

### Approvals

| **Role** | **Notes** | **Name / Title** | |
| --- | --- | --- | --- |
| **Peer Review** |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **Oversight Review** |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **Stakeholder Review** |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Sign Off

| **Role** | **Name** | **Title** | **Date** | **Signature** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |