**Software Design Specification**

**Document**

**[Document Template Management System]**

**Version: (1.0.0)** **Date: (01/24/2017)**

**Table of Contents**

1 Introduction 1

1.1 System Overview 1

1.2 Definitions, Acronyms, and Abbreviations 1

1.3 References 1

1.4 Document Map 1

2 Design Considerations 2

2.1 Assumptions 2

2.2 Constraints 2

2.3 System Environment 2

2.4 Design Methodology 2

3 Architectural (High-level) Design 3

3.1 Overview 3

3.2 Rationale 3

3.3 Conceptual (or Logical) View 3

3.4 Other Views 3

4 Low Level Design 4

4.1 Component 1 4

4.2 Component 2 4

4.n Component n 4

5 User Interface Design 5

5.1 Application Control 5

5.2 Screen 1 5

5.3 Screen 2 5

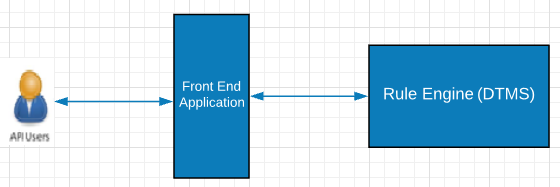
5.n Screen n 5

# Introduction

The following subsections of the Software Design Specifications (SDS) document provide an overview of Document Template Management System. Document Template Management system will be used to fetch appropriate templates (Mail/SMS/Physical Letter) for different processes and scenarios based on various rules that would be configured in Decision Table of Business Rules Engine. These templates will be modified to replace the dynamic fields with their actual values at run time and the updated template will be sent to the consuming application. Business users will be able to add new rules, update existing rules and view the existing rules using the Front end UI portal.

## System Overview

**Consumption Flow:**



1. A user will request a template from the front end portal. The portal will send a request to Rule Engine (Document Template Management system).
2. Document Template Management system

## Definitions, Acronyms, and Abbreviations

| **Term/Acronym** | **Definition** |
| --- | --- |
| CAPIOT | CAPIOT Software Pvt. Ltd, the proposed vendor of services in this proposal |
| Cnk | Cox and Kings |
| DTMS | Document Template Management System |
| ESB | Enterprise service bus |
| MDM | Master Data Management |
| BE | Booking Engine |
| SI | Supplier Integration |

## References



## Document Map

Define all major sections of this document and provide a one-sentence summary of each.

# Design Considerations

These subsections describe issues that need to be addressed or resolved prior to or while completing the design as well as issues that may influence the design process.

## Assumptions

* + 1. All the dynamic fields present in the template will be received in the input to DTMS Rule Engine. If the dynamic fields are not received in the input then those fields will be replaced with blank character.
    2. The template will be sent as HTML content to the consuming application.

## Constraints

2.2.1. The HTML output shall conform to the HTML standard.

2.2.2. All the business rules will be written in BRMS rule language.

## System Environment

**2.3.1. Hardware Requirements:**

**2.3.2. Software Requirements:**

* BRMS 6.5
* Apache Ant 1.10.1
* Java (Jdk 1.8)
* Eclipse

## Design Methodology

Summarize the approach that will be used to create and evolve the design for this system. This is not a rehash of your project lifecycle or change-management plan. This is for stating whether you will use object-oriented design, formal specifications, or other specific methodologies. Most people will use some object-oriented technique with UML.

# Architectural (High-level) Design

**The architecture provides the top-level design view of a system and provides a basis for more detailed design work. These subsections describe the top-level components of the system you are building and their relationships. For an OO implementation in Java, for example, our components could become packages (or set of packages, depending on the level of granularity considered and the size of the system).**

**In defining your architectural design, you can follow one of the organizational styles seen in class (shared data repository, shared services and servers, and abstract machine/layered) or pick a different one if none of those is appropriate for your system.**

## Overview

This section provides a high level overview of the structural and functional decomposition of the system.

## Rationale

This section discusses why you are using the architecture you have chosen.

## Conceptual (or Logical) View

This section should provide and describe a diagram that shows the various components and how they are connected. The conceptual view shows the logical/functional components of the system, where each component represents a cluster of related functionality. For UML, this would typically be a component diagram or a package diagram.

## Other Views

**High-level designs are most effective if they attempt to model groups of system elements from a number of different views. Beside the Conceptual/Logical view, examples of additional viewpoints are:**

1. **Process View: this represents the runtime view of the system. The components are threads, processes, or distributed components. In UML, this would typically be a process interaction diagram.**
2. **Physical View: this view is for distributed systems. The components are physical processors that have parts of the system running on them. For UML, this would be a deployment diagram.**

Note that it is not necessary to document all these views. For many smaller applications, the conceptual view is all that is necessary. Document those views that will help you design and implement the system and create a subsection for each one of them.

# Low Level Design

This section provides the low-level design for each of the system components identified in the previous section. For each component, you should provide a subsection that shows its internal structure. In the case of an OO design, this internal structure would typically be expressed as an UML class diagram that represents the static class structure for the component. For smaller systems, you may have a single UML class diagram that each component description refers to.

## Component 1

## Component 2

## 4.n Component n

As discussed above, these subsections should provide and discuss detailed diagrams of each software module. For at least some of the components, you should provide diagrams that show a dynamic view of the component internals (i.e., that show the dynamic interaction between classes). In the case of an OO design, UML state or interaction diagrams can be used to this end.

# User Interface Design

**These subsections discuss the user interface design.**

## Application Control

This section details the common behavior that all screens will have. Common look and feel details, such as menus, popup menus, toolbars, status bars, title bars, drag and drop, and mouse behavior should be described here.

## Screen 1

## Screen 2

## 5.n Screen n

These sections illustrate all major user-interface screens and describe the behavior and state changes that the user will experience. A screen transition diagram or table can optionally be created to illustrate the flow of control through the various screens.

Note that these sections may not show actual screenshots (in case you have not completed the implementation yet). In these cases, they can be drawings or mockups created using some rapid GUI-building tool.