**Software Design Specification**

**Document**

**[Commercials Calculation Engine]**

**Version: (1.0.0)** **Date: (01/22/2017)**

**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

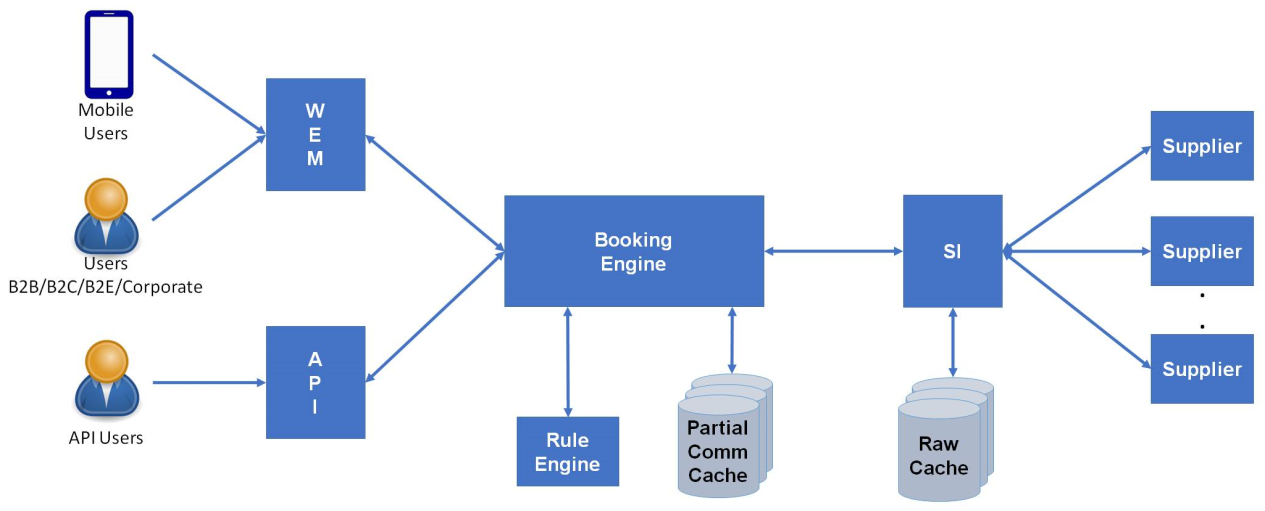
5.n Screen n 5

# Introduction

**The following subsections of the Software Design Specifications (SDS) document should provide an overview of the entire SDS. The thing to keep in mind as you write this document is that you are telling how the system should do what it is supposed to do, so that the system can be implemented.**

## System Overview

**Consumption Flow :**



1. A user will request a product (flights, buses, hotels etc.) search from user portal or through API. This request will be routed to Booking Engine (BE) by Web Experience Management (WEM) layer or API layer.
2. The Booking Engine will check if a similar search was performed by the client previously and whether the results of this previous search are available in Commercials cache.
   1. If the results are found in Commercials cache, advanced filtering for current search (if any) will be applied on the retrieved cached results. These filtered and enriched search results will be returned to WEM for rendering to the user.
   2. If the Booking Engine does not find search results in cache, the request will be forwarded to Supplier Integration (SI) layer.
3. The SI layer will have its own atomic raw cache. The SI services will check if the same search was performed previously and whether the results of this previous search are available in the atomic raw cache.
   1. If the results are found in atomic raw cache, the results will be retrieved from cache and returned to the Booking Engine for further processing.
   2. If the results are not available in cache, the request is sent to various suppliers. The search results in responses received from these suppliers are cached for future use, consolidated together and sent to the Booking Engine for further processing.
4. When Booking Engine receives results from SI layer, it will apply advanced filtering for current search (if any).
5. The result is then sent to Commercials Calculation Engine. Commercials Calculation Engine applies various configured Supplier commercials (e.g. Standard Commercial, Overriding Commercial, and Productivity Linked Bonus etc.) and sends back the response to Booking Engine.
6. Booking Engine then enriches the supplier commercials response with client context information like client entity type, client entity name etc. and sends the request to Commercials Calculation Engine to apply Client commercials. After applying Client commercials, CCE sends back the response to Booking Engine.
7. Booking Engine sends the filtered and enriched search results to WEM for rendering to the user or to API users.
   1. Booking Engine will store the search results in Commercials Cache for future use.
8. The user will browse through the results and may select a result to look at details of that result. The WEM/API layer will request price verification for the selected search result to the Booking Engine.
9. The Booking Engine will forward this price verification (re-price) request to SI layer.
10. The SI layer will call supplier service to fetch the latest price for selected product. The latest price will be returned to Booking Engine for further processing. The latest price will also be updated in SI side atomic raw cache.
11. The Booking Engine will receive the latest price, send request to Commercials Calculation Engine to perform all commercial calculations (supplier & client specific commercials) and return the results of price verification to WEM for displaying it to the user or to API users. The Booking Engine will update its Commercials Cache with the latest price information.

**Configuration Flow :**

## Definitions, Acronyms, and Abbreviations

| **Term/Acronym** | **Definition** |
| --- | --- |
| CAPIOT | CAPIOT Software Pvt. Ltd, the proposed vendor of services in this proposal |
| Cnk | Cox and Kings |
| CCE | Commercials Calculation Engine |
| ESB | Enterprise service bus |
| MDM | Master Data Management |
| OMS | Order Management System |
| ODS | Operational Data Store |
| 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

2.1.1. If there are conflicting rules being configured in the UI, then Business needs to set the priority of the conflicting rules and send to CCE.

E.g.: Rule 1: Travel date -"5-07-2017 to 5-09-2017"

Rule 2: All conditions same as Rule 1 except Travel date which is "20-07-2017 to 31-07-2017". If rule 2 has to be executed, then UI will send higher priority for Rule2.

2.1.2. If the supplier commercial is configured on fare component as “Base Fare +Tax1”, then CCE expects Base fare and Tax1 to be present in the input coming from Booking Engine. If any component (e.g. Tax1) is not present in the input, then commercials will not be applied on that component.

2.1.3. Currency Conversion and tax calculation will be out of scope for CCE.

2.1.4. In Packages, CCE will receive total base fare (sum total base fare of all the included products like Acco, Insurance etc.) , total taxes (sum total taxes of all the included products like Acco, Insurance etc.), total surcharge, total discount etc. from Booking Engine for set packages.

2.1.5. Attaching policy is not being handled currently in CCE.

2.1.6. In case of the commercials which are payable to the supplier (e.g. Service charge), the company will have a retention of minimum 100% while applying the client commercials so that the company does not bear the losses.

2.1.7. Client commercials: In fixed model, even if we are not getting a supplier commercial (E.g.: PLB), still if configured at client side, client may apply a fixed amount/percentage on PLB.

Supplier commercials: Slab values [e.g.: Total turnover] will not be checked for commercials during the transactional flow. Slab values will be checked and applied when the actual settlement will happen.

2.1.8. Markup (Client commercials): Cnk Company has a client "Akbar Travels" and "Akbar Travels" has a client "ABC". If Cnk applies a markup, it is applied on the passenger fare.

Now if "Akbar Travels" doesn’t apply any markup but "ABC" applies a markup, then it will be applied on the fare calculated by Cnk and not on the passenger fare.

## Constraints

Describe any constraints on the system that have a significant impact on the design of the system. (e.g., technology constraints, performance requirements, end user characteristics) These are things the customer has told you that directly influence the design (e.g., the DB must be an open-source, freely available DBMS).

## System Environment

<<Describe the hardware and software that the system must operate in and interact with. >>

**Software Requirements:**

Apache Ant 1.10.1

Java (Jdk 1.8)

BRMS 6.5

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. The section should list the different components and concisely discuss the major responsibilities and roles such components must play.

## 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.