**Design document**

***Configurator Project***

*Copaco Ltd.*

A logo of hands and flowers

Description automatically generated

|  |
| --- |
| **Date : 31.10.2024** |
| **Version : 1.0** |
| **State : Draft** |
| **Author : CoLab Team** |

#### Version history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 0.1 | 31.10.2024 | CoLab Team | Initial draft | Draft |

**Distribution**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Receivers** |
| 1 | 31.10.2024 | Copaco Ltd. |

Contents

[1. Introduction 4](#_Toc181282225)

[2. Architecture constraints 5](#_Toc181282226)

[3. C4 Model Diagrams 6](#_Toc181282227)

[3.1 C1 diagram 6](#_Toc181282228)

[3.2 C2 diagram 7](#_Toc181282229)

[3.3 C3 diagram 9](#_Toc181282230)

# Introduction

The purpose of this document is to outline the design and architecture of the *Configurator* software project. The project aims to define a software solution for the client, *Copaco Ltd.*, whose wish is to extend their e-commerce environment and provide a new feature for their clients.

The following chapters contain the reasoning behind the decisions that were made for structuring this project, in order for the final product to prove reliability, scalability and quality. Therefore, this paper provides a comprehensive explanation of how the modern technologies and design strategies are combined so that the delivered system satisfies the client with a great experience.

# Architecture constraints

The system is build through a full-stack approach, following a set of constraints provided by the client, while also taking into consideration personal preferences and opinions:

* *React* (frontend)

React provides a component-based architecture for building a user interface in a single page application. This ensures a smooth experience for the users and high performance in communication with the backend. Although the client was previously using *Angular,* the team opted for a language in which they are experienced, therefore ensuring a better quality of the final product.

* *Spring Boot + Java* (backend)

Spring Boot, a Java-based framework, allows for a rapid development of the backend application and configuration of a RESTful API that communicates with the frontend.

* *MSSQL*(database layer)

The database is designed so that redundancy is reduces, providing an architecture that optimizes the performance of the application. Since the communication with the backend is done through JDBC, connection handling and queries management ensure smooth data transfers. This database is also used by the client for their current platform.

# C4 Model Diagrams

This chapter contains diagrams that describe the architecture of the application, starting with a high level view, and then diving into a detailed visual representation of its structure.

## C1 diagram

A screenshot of a computer screen

Description automatically generated

This C1 diagram represents the main actors and system involved in the **Copaco Configurator** project.

1. **Admin**:
   * Role: Person
   * Description: The Admin is a user with permissions to manage the templates and rules for the configurator. This role is responsible for defining and maintaining the configuration templates and rules that will be used within the Copaco Configurator.
2. **Customer**:
   * Role: Person
   * Description: The Customer is a user who can create a personal account and access the configurator. This role allows individuals to utilize the configurator to design and view configurations, as well as access lists of available templates and their previous configurations.
3. **Copaco Configurator**:
   * Role: Software System
   * Description: The core system that enables customers to use the configurator functionality defined by admins. The configurator provides access to a list of available templates and allows customers to view previous configurations. It bridges the interaction between the Admin's defined configuration rules and the Customer's use of the configurator to design products.

## C2 diagram

A diagram of a diagram

Description automatically generated

This C2 (Container) diagram provides a detailed view of the **CareNest** platform’s architecture, showcasing the interactions between the main components and their roles.

1. **Admin**:
   * Role: Person
   * Responsibilities: The Admin user can view available templates, configure products, and manage previous configurations. This role manages the template components and the rules that guide configuration.
2. **Customer**:
   * Role: Person
   * Responsibilities: The Customer user has a personal account to interact with the configurator. They can manage and use components within templates based on the rules defined by the Admin.
3. **Single Page Application (SPA)**:
   * Container: React & Spring Boot
   * Functionality: The SPA serves as the front-end interface, providing functionalities accessible through the web browser. Both Admins and Customers interact with this layer to perform actions such as viewing templates and configurations.
   * Interaction: The SPA makes API calls to the back-end web application for data operations.
4. **Web Application**:
   * Container: Java Spring Boot (REST API)
   * Functionality: This is the back-end layer of the system, responsible for processing business logic and handling data manipulation. It acts as the intermediary between the SPA and the database, ensuring data consistency and enforcing the rules defined by the Admin.
   * Interaction: The web application handles API requests from the SPA and interacts with the database to retrieve and store data.
5. **Database**:
   * Container: MSSQL
   * Functionality: The database stores essential data for the platform, including user information, announcements, and requests.
   * Interaction: The Web Application reads from and writes to this database to manage user accounts, configurations, and other relevant data.

## C3 diagram

A screenshot of a computer

Description automatically generated

This C3 diagram provides an overview of the **Copaco Configurator application architecture**, detailing the interactions between the **Controller Layer**, **Business Layer (Services)**, and **Persistence Layer (Repositories)**. Here’s a summary of each layer:

1. **Controller Layer**:
   * This layer consists of controllers for different functional areas, handling incoming HTTP requests from the frontend. Each controller forwards requests to the respective service for processing.
   * Controllers manage high-level application operations such as user management, product configuration, and compatibility checks.
2. **Business Layer (Services)**:
   * The services are responsible for business logic and interact with multiple repositories to retrieve and manipulate data.
   * Each service corresponds to a specific domain (e.g., UserService, ProductTemplateService) and coordinates actions across the persistence layer to fulfill application needs.
   * The services enforce application rules, such as product template rules, component compatibility, and user role verification, ensuring data consistency and integrity.
3. **Persistence Layer (Repositories)**:
   * This layer contains repository classes responsible for direct data access and database operations.
   * Each repository manages a specific table or entity (e.g., UserRepository, ComponentRepository), abstracting the data access logic from the business layer.
   * Repositories interact with each other where necessary, such as the CompatibilityRepository depending on ComponentRepository and ComponentTypeRepository for compatibility checks.
4. **Database**:
   * The repositories communicate with the database, storing and retrieving data for entities like users, components, templates, and compatibility rules.
   * The data model supports modular configuration, allowing for customizable products and compatibility management.

# API Calls diagram

## Template Controller

### GET

* “/templates”
  + No parameters
  + Response:

[

{

"templateId": 0,

"category": {

"categoryId": 0,

"categoryName": "string"

},

"name": "string",

"brand": {

"id": 0,

"name": "string"

},

"imageUrl": "string",

"components": {

"additionalProp1": 0,

"additionalProp2": 0,

"additionalProp3": 0

}

}

]

* “/templates/{id}”
  + id – integer(required)
  + no request body
  + response:

{

"templateId": 0,

"category": {

"categoryId": 0,

"categoryName": "string"

},

"name": "string",

"brand": {

"id": 0,

"name": "string"

},

"imageUrl": "string",

"components": {

"additionalProp1": 0,

"additionalProp2": 0,

"additionalProp3": 0

}

}

* “/templates/name/{name}”
  + name – string (required)
  + no request body
  + response:

[

{

"templateId": 0,

"category": {

"categoryId": 0,

"categoryName": "string"

},

"name": "string",

"brand": {

"id": 0,

"name": "string"

},

"imageUrl": "string",

"components": {

"additionalProp1": 0,

"additionalProp2": 0,

"additionalProp3": 0

}

}

]

### POST

* “/templates”
  + No parameters:
  + Request body:

{

"categoryId": 0,

"name": "string",

"brandId": 0,

"imageUrl": "string",

"componentTypes": [

{

"componentTypeId": 0,

"orderOfImportance": 0

}

]

}

* + Response: http status

### DELETE

* “/templates/{id}”
  + id – integer
  + no request body
  + response : http status

## 4.2 Component type

### GET

* “/componentType”
  + No parameters
  + No request body
  + Response:

{

"componentTypes": [

{

"componentTypeId": 0,

"componentTypeName": "string",

"componentTypeImageUrl": "string",

"category": {

"categoryId": 0,

"categoryName": "string"

},

"specificationTypeList": [

{

"specificationTypeId": 0,

"specificationTypeName": "string"

}

]

}

]

}

* “/componentType/{id}”
  + Id – integer (required)
  + No request body
  + Response:

{

"componentTypeId": 0,

"componentTypeName": "string",

"componentTypeImageUrl": "string",

"category": {

"categoryId": 0,

"categoryName": "string"

},

"specificationTypeList": [

{

"specificationTypeId": 0,

"specificationTypeName": "string"

}

]

}

## Component Controller

### GET

* “/components”
  + No parameters
  + No request body
  + Response :

[

{

"componentId": 0,

"componentName": "string",

"componentTypeId": 0,

"componentTypeName": "string",

"categoryName": "string",

"componentImageUrl": "string",

"brandName": "string",

"componentPrice": 0,

"specifications": {

"additionalProp1": [

"string"

],

"additionalProp2": [

"string"

],

"additionalProp3": [

"string"

]

}

}

]

* “/components/{categoryId}”
  + categoryId – integer
  + no request body
  + Response:

[

{

"componentId": 0,

"componentName": "string",

"componentType": {

"componentTypeId": 0,

"componentTypeName": "string",

"componentTypeImageUrl": "string",

"category": {

"categoryId": 0,

"categoryName": "string"

},

"specificationTypeList": [

{

"specificationTypeId": 0,

"specificationTypeName": "string"

}

]

},

"componentImageUrl": "string",

"brand": {

"id": 0,

"name": "string"

},

"componentPrice": 0,

"specifications": {

"additionalProp1": [

"string"

],

"additionalProp2": [

"string"

],

"additionalProp3": [

"string"

]

}

}

]

## Category controller

### GET

* “/category”
  + No Parameters
  + No request body
  + Response:

[

{

"categoryId": 0,

"categoryName": "string"

}

]

## Specification type controller

### GET

* “/specificationType”
  + No parameters
  + No request body
  + Response:

{

"specificationTypes": [

{

"specificationTypeId": 0,

"specificationTypeName": "string"

}

]

}

* “/specificationType/id/{id}”
  + Id – integer (required)
  + No request body
  + Response:

{

"specificationTypeId": 0,

"specificationTypeName": "string"

}

* “/specificationType/componentId/{componentId}”
  + componentId – integer (required)
  + No request body
  + response:

[

{

"specificationTypeId": 0,

"specificationTypeName": "string"

}

]

### POST

* “/specificationType”
  + Parameters:

{

"specificationTypeName": "string",

"specificationTypeId": 0

}

* + No request body
  + Response: http status

## Compatibility controller

### GET

* “/compatibility”
  + No Parameters
  + No request body
  + Response:

[

{

"id": 0,

"typeOfCompatibility": "string"

}

]

* “/compatibility/getAllComponentsFromAGivenComponentTypeAndSpecification”
  + Parameters:
    - componentId: integer(required)
    - searchedComponentsType : integer (required)
  + No request body
  + Response:

[

{

"componentId": 0,

"componentName": "string",

"componentType": {

"componentTypeId": 0,

"componentTypeName": "string",

"componentTypeImageUrl": "string",

"category": {

"categoryId": 0,

"categoryName": "string"

},

"specificationTypeList": [

{

"specificationTypeId": 0,

"specificationTypeName": "string"

}

]

},

"componentImageUrl": "string",

"brand": {

"id": 0,

"name": "string"

},

"componentPrice": 0,

"specifications": {

"additionalProp1": [

"string"

],

"additionalProp2": [

"string"

],

"additionalProp3": [

"string"

]

}

}

]

* “/compatibility/automaticCompatibility/{automaticCompatibilityId}”
  + automaticCompatibilityId: integer (required)
  + No request body
  + Response:

{

"automaticCompatibilityId": 0,

"componentType1Id": 0,

"componentType2Id": 0,

"specificationTypeFromComponentType1": {

"specificationTypeId": 0,

"specificationTypeName": "string"

},

"specificationTypeFromComponentType2": {

"specificationTypeId": 0,

"specificationTypeName": "string"

}

}

* “/compatibility/allAutomaticCompatibilitiesByGivenComponentTypeId/{componentTypeId}”
  + componentTypeId: integer(required)
  + No request body
  + response:

[

{

"automaticCompatibilityId": 0,

"componentType1Id": 0,

"componentType2Id": 0,

"specificationTypeFromComponentType1": {

"specificationTypeId": 0,

"specificationTypeName": "string"

},

"specificationTypeFromComponentType2": {

"specificationTypeId": 0,

"specificationTypeName": "string"

}

}

]

### POST

* “/compatibility/automatic”
  + No parameters
  + Request:

{

"componentType1Id": 0,

"componentType2Id": 0,

"specificationToConsiderId\_from\_component1": 0,

"specificationToConsiderId\_from\_component2": 0

}

* + Response: http status