FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Application for configuration of modular products
Student:	Michal Dobeš
Supervisor:	Ing. Jiří Hunka
Study program:	Informatics
Branch / specialization:	Business Informatics 2021
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2024/2025

Instructions

Cílem této práce je návrh, realizace a následné prototypové (ukázkové) nasazení aplikace umožňující zákazníkům online 3D konfiguraci modulárních produktů. Důraz je kladen na univerzální využití daného konfigurátoru nezávisle na cílovou oblast použití.

Postupujte v těchto krocích:
1. Proveďte důkladnou analýzu s ohledem na potenciální cílový okruh budoucích uživatelů. Zaměřte se také na již existující řešení.
2. Na základě analýzy vytvořte vhodný návrh.
3. Na základě návrhu vytvořte použitelnou aplikaci.
4. Výsledný prototyp řádně otestujte vhodně zvolenými testy.
5. Analyzujte dopad implementace aplikace na business procesy vybrané společnosti. Využijte ukázkově vaše řešení právě pro vámi zvolenou společnost.
6. Zhodnoťte dosažené výsledky a navrhněte možná budoucí vylepšení.

Electronically approved by Ing. David Buchtela, Ph.D. on 22 January 2024 in Prague.

Bachelor's thesis

APPLICATION FOR CONFIGURATION OF MODULAR PRODUCTS

Michal Dobeš

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Jiří Hunka
May 15, 2024

Citation of this thesis: Dobeš Michal. *Application for configuration of modular products.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

List of Code Listings

I want to express my heartfelt gratitude to my supervisor, Ing. Jiří Hunka, for enabling me to create a bachelor's thesis on this topic and for providing invaluable advice throughout the entire process.

I am also profoundly thankful to the testers for their time and feedback in the usability testing of the solution.

Last but not least, I want to thank my family and friends for their inexhaustible moral and material support, which enabled my bachelor's study at the university.

Declaration

In Prague on May 15, 2024

Abstract

This bachelor's thesis focuses on the design and development of a web application for 3D online configuration of modular products. The thesis examines comparable existing applications and implements a product-agnostic, front-end-only, customizable web configurator conceived for use by small businesses. The configurator provides a straightforward definition of configurable products and operates solely using a web server. The implementation leverages modern technologies, combining Three.js with React. The thesis furthermore describes an example deployment of the created solution in a real-world scenario and discusses the business aspects of the application. The result of this bachelor's thesis is a tool businesses can use to introduce modular product configuration on their websites.

Keywords mass customization, 3D configurator, web application, modular product, Three.js, React, front-end development

Abstrakt

Tato bakalářská práce se zabývá návrhem a vývojem webové aplikace pro 3D online konfiguraci modulárních produktů. Práce analyzuje existující srovnatelné aplikace a implementuje produktově nezávislý, pouze front-endový a přizpůsobitelný webový konfigurátor koncipovaný pro použití v malých podnicích. Konfigurátor umožňuje přímočarou definici konfigurovatelných produktů a funguje výhradně pomocí webového serveru. Implementace využívá moderní technologie a kombinuje Three.js a React. Práce dále popisuje nasazení vyvinutého řešení na příkladném scénáři v reálném prostředí a jsou diskutovány byznysové aspekty aplikace. Výsledkem této bakalářské práce je nástroj, který mohou podniky využít k zavedení modulární konfigurace produktů na svých webových stránkách.

Klíčová slova masová kustomizace, 3D konfigurátor, webová aplikace, modulární produkt, Three.js, React, vývoj front-endu

List of Abbreviations

API	Application Programming Interface
AR	Augmented Reality
BVH	Bounding Volume Hierarchy
CAD	Computer-Aided Design
CD	Continuous Deployment
CI	Continuous Integration
CSS	Cascading Style Sheets
DNS	Domain Name System
DOM	Document Object Model
ERP	Enterprise Resource Planning
FCP	First Contentful Paint
GLB	Graphics Library Transmission Format Binary
gLTF	Graphics Library Transmission Format
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
PDF	Portable Document Format
R3F	React Three Fiber
ROI	Return on Investment
SPA	Single-Page Application
SUS	System Usability Scale
UML	Unified Modeling Language
URL	Uniform Resource Locator
USDZ	Universal Scene Description Zip
VR	Virtual Reality
WebGL	Web Graphics Library

Introduction

Product configurators and their value.

Over the past few decades, the rise of e-commerce has caused a shift in consumer expectations, resulting in an increased demand for individualized products. This gives rise to the need to shift focus towards mass customization, where products are customized according to individual preferences. To thrive in this sector, companies must modify their product offerings to meet the unique needs of users. This necessitates the existence of a system that enables customers to express their preferences and convert them into product configurations. [1]

The task of transforming user preferences into concrete designs is a difficult endeavor that can be hindered by a lack of effective communication between the customer's explanation of their desires and the business's comprehension. The use of online product configurators seemingly provides a solution to this issue by offering a user-friendly and visually appealing platform that allows customers to customize products according to their needs, improves the customer experience by increasing engagement and interactivity, and helps bridge the gap between customer expectations and the end product. These tools have become an integral part of successful personalization strategies. [2]

The involvement of consumers in the customization process leads to a stronger bond with the product compared to standard off-the-shelf products. This aspect of mass customization makes it an appealing and compelling strategy for businesses to implement [3]. However, when implementing such a system, it is crucial to ensure that the customization process is pleasurable for the customer. Research has shown that the enjoyment experienced during the customization process also affects the perceived value of the final product, highlighting the importance of a good implementation. [4]

The introduction of modern technologies such as WebGL or Augmented Reality (AR) has expanded the potential of online configurators. These advances enable these toolkits to become more powerful and visually illustrative tools that provide a higher level of interactivity and realism than what was previously accessible. [5]

Objective of This Thesis

The primary objective of this thesis is to design and implement an application (toolkit) for the 3D online configuration of modular products. The toolkit aims to be product-agnostic, adaptable, and customizable, usable by various businesses, enabling their customers to customize their modular products interactively. The focus is on ensuring that the toolkit is not only flexible in accommodating various specific needs, but also straightforward for businesses to maintain after deployment, emphasizing lightweight infrastructure requirements.

To accomplish this main objective, an accompanying analysis of the characteristics found in current product configurators is required, as well as an examination of comparable solutions currently available to businesses.

Structure of This Thesis

This thesis is divided into six chapters.

Chapter 1 The initial chapter entails an examination of existing solutions and an investigation into the functionalities that should be incorporated into this particular application.

Chapter 2 The second chapter discusses the design of the application and the technologies chosen.

Chapter 3 The third chapter is devoted to implementation.

Chapter 4 Chapter four focuses on the example deployment of the implemented application in a chosen example business. In addition, it discusses the resulting changes in the business processes of a particular business.

Chapter 5 In the fifth chapter, the tests used in the development of the application are described.

Chapter 6 Finally, the last chapter summarizes the results achieved and suggests possible directions for future development.

Chapter 1
Analysis

A comprehensive evaluation of existing solutions, identification of key features and limitations, and outlining of the specifications for the proposed solution.

Product configurators can be implemented in various ways, and the design of the tool itself determines the types of products that can be customized using the tool later on. The number of unique configurations of a product that the tool can create is called the solution space. The size of the solution space is determined by the count of customizable attributes and the achievable values of each attribute [6]. A relevant study [7] examined the solution spaces of these toolkits and proposed an evaluation model that enables the categorization and assessment of various implementation approaches. Based on the target outcome and the guidance provided by the tool, the following mechanisms are defined:

▶ **Definiton 1.1** (Veneer)**.** *Customization by adding a visual decorative layer. (e.g., printing, engraving, etching)*

▶ **Definiton 1.2** (Modularity)**.** *Customization by combining modules or components.*

▶ **Definiton 1.3** (Parametric)**.** *Customization by changing the parameter values of parts.*

▶ **Definiton 1.4** (Generative)**.** *Customization using code and scripting to synthesize the final form of the product.*

There are often some common characteristics among configuration tools with different mechanisms; however, the main focus of this thesis is on toolkits that primarily employ modularity mechanisms.

1.1 Existing Solutions

1.1.1 Applications of Product Configurators

Many companies across multiple industries, such as automotive, fashion, furniture, housing, are integrating product configurators into their sales strategies. These configurators serve as the main or supplementary sales tools for these businesses.

The Configurator Database Project by cyLEDGE MEDIA aims to catalog these web-based configuration tools. The 2017/2018 report tracked 1250 deployments of these tools; however, the true count will be significantly higher since the database only includes the most frequently visited applications. [8]

An analysis of the 100 most viewed configurators from May 2020 to May 2021 in the Configurator Database Project was performed in a study [9] that examined the shared characteristics of these configurators. The summary of some of the relevant characteristics and design choices that the study has analyzed are presented in this section:

Responsive design: 75.3% of examined tools had responsive design (the design adapted to the viewport of the device).

Navigation: 17.5% of configurators had linear predefined navigation (meaning the configuration had to follow a specified sequence), whereas the 82.5% majority of tools had open navigation (meaning the user has the flexibility to configure the product in any order).

Visualization: 79.4% of tools utilized photorealistic visualization (as opposed to illustrations or no visualization); however, the study acknowledges that there were significant variations depending on the industries in which the configurator is utilized.

Data transfer: The mean network data size transferred for 3D configurator was 35.6 MB.

Configuration options: 60.8% of configurators offered more than ten customizable attributes.

Purchase capability: Given that car brands typically do not directly sell their cars online, they were excluded from the analysis of this particular characteristic. Without vehicle configurators, 70.5% of the configurators could complete an online purchase of the configured product.

Price calculation: 56.7% of the configurators were able to instantly reflect the changes made to the configuration in the displayed price.

The previous paragraphs discussed general trends among all product configurators of all kinds. As part of the analysis in this chapter, it is essential also to examine existing particular modular 3D product configurators.

Due to the large number of existing applications, it is not within the scope of this work to perform an exhaustive analysis. Instead, this section will focus on a select group of three applications. These have been selected based on a combination of factors such as their popularity, functionality, and importance in the context of a modular product configuration. This selection is intended to provide insightful examples that highlight different approaches, rather than being representative of the entire domain.
The main aspects under consideration are as follows:

- **Platform:** How is the application accessible?
- **Navigation:** How does the user navigate in the application during the configuration process?
- **Visualization style:** How is the product visualized within the configuration process?
- **Placement options:** Can the modular components be freely placed, or are they restricted to fixed points?
- **Camera movement:** From which angles can the product be visualized, and how can it switch between them? In the context of the tool, camera refers to a virtual camera in a 3D graphical environment that simulates the viewpoint and perspective from which the 3D scene is viewed.
- **Impossible configurations:**Is it possible to create configurations that are not feasible in reality, such as physically overlapping components?
- **Responsiveness:** How does the application adapt to different device viewports?
- **Price calculation:** Is the price of the configured product calculated in real-time?
- **Purchase option:** Is there an option to finalize and purchase the configured product within the application?
- **Save option:** Can users save their configurations to return to them later?
- **Version history:** Does the configurator provide an accessible history of configuration changes?
- **VR or AR:** Can the configured product be visualized in Virtual Reality or Augmented Reality?
- **Real dimensions**: Can the configurator display information about the real dimensions of the configured products?

Furthermore, the design decisions are also discussed:

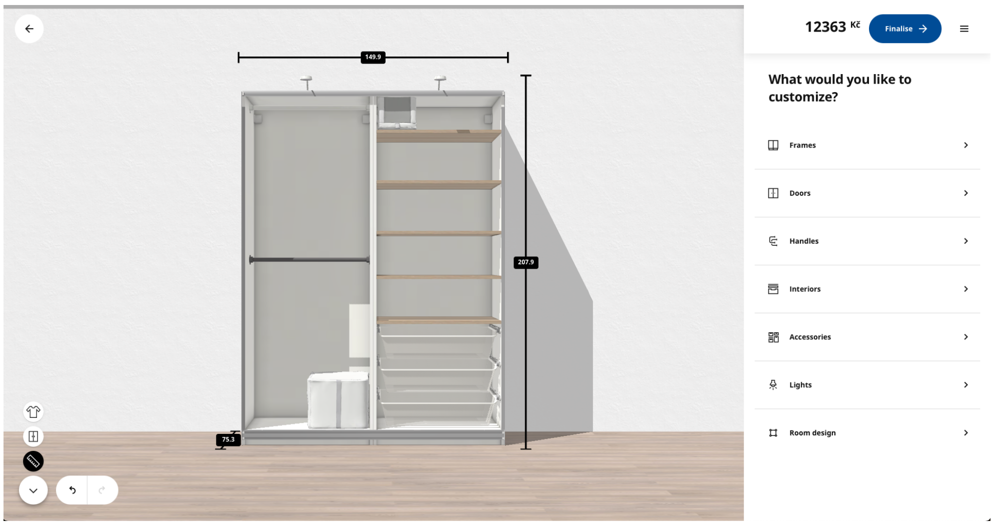
- **Views:** What is the position and size of the views inside the application?
- **Navigation bars:** Where are the navigation bars placed and how are they utilized?
- **Button placements:** How are different buttons placed within the application's interface?

1.1.1.1 IKEA PAX Planner

IKEA is a widely recognized global home furnishings retailer specializing in affordable furniture. [10]

IKEA offers the PAX fitted wardrobe, for which they not only sell predefined configurations, but also allow customers to modularly choose the ideal size, doors, knobs, handles, interior organization, and lightning. [11]

To accomplish this, they utilize the PAX Planner web tool.[1]



■ **Figure 1.1** Screenshot of IKEA PAX Planner Tool with example configuration
Source: IKEA [11]

The tool consists mainly of two views. The primary view on the left contains a 3D preview of the configured product, allowing users to observe objects from different viewpoints by moving along an orbital trajectory in a 180-degree half-circle. The components displayed in the 3D view are both realistic and interactive. Users can adjust their position by dragging, and selecting a component offers additional information along with real-life images of the item. All modular options that can be added to the current configuration

[1]Available at: `https://www.ikea.com/addon-app/storageone/pax/web/latest/cz/en/`

are found in the secondary smaller view on the right side and can be added by clicking or dragging them into the 3D preview. The application is responsive, and on mobile devices, the secondary view moves from the right side to a bottom sliding panel. The navigation bar is located at the top of the secondary view, and other buttons are located around the edges of the primary view.

At the beginning of the configuration process, the application prompts the user to select the starting point of the configuration. The configurator has open navigation, meaning that the components can then be configured in any order. Components can be placed anywhere along a specified axis with certain restrictions, effectively preventing the creation of impossible configurations.

The tool performs live price calculations and contains a final summary confirmation screen from which it is possible to order the configured product in the e-shop. The configurator maintains a history of recent changes, accessible through undo and redo buttons. It also features the ability to save configurations on the server, which can be retrieved later using a generated code.

The configurator also provides a range of innovative features, such as the ability to change the visibility of some elements using a button (e.g. hiding the doors of a wardrobe to reveal the contents inside) or the ability to display dimensions directly in the 3D preview.

The application is a Single-Page Application (SPA) and does not update the URL based on the selected product or phase of the configuration.

SPA is a web application implementation approach that loads only a single page and then sequentially updates the content of the page with scripting on the client side, rather than loading whole new pages from the server. [12]

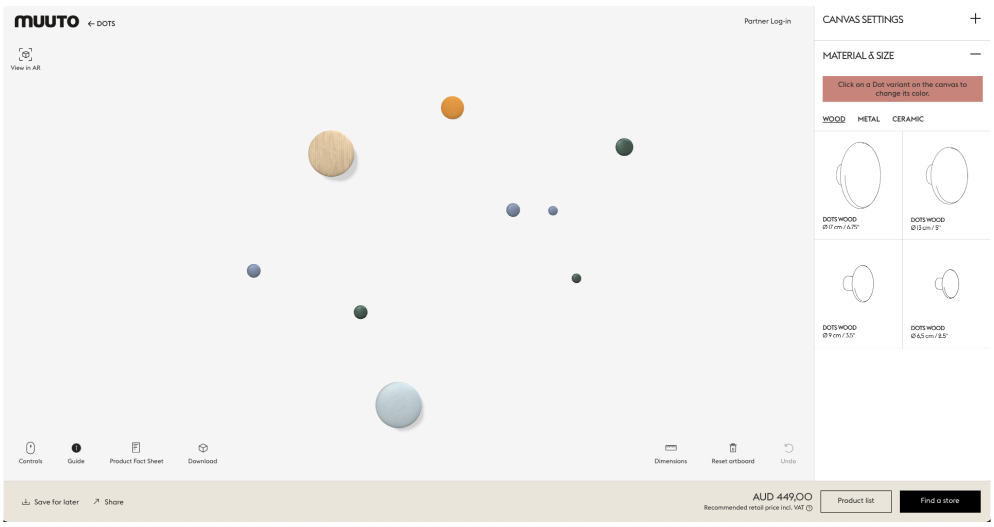
1.1.1.2 Muuto Product Planner

Muuto is a Scandinavian design company that produces furniture and home accessories. [13]

The company provides Product Planner, a 3D web-based configurator, which allows customers to customize and combine the designs of various products, such as storage systems, sofas, tables, or wall hangers, tailored to their specific needs.[2] [14]

The design of the configurator is similar to that in the previous case. The configurator also consists of two views. The primary view on the left provides full realistic 3D visualization, while the smaller secondary view on the right side allows users to add components by dragging them into the main view. Selecting a component in the primary view enables users to remove it or alter its materials. As the design is responsive, on smaller devices, the secondary view transforms into a bottom slide panel. Depending on the configured product, the tool offers a preview either from a single angle or a preview from any point on an orbital trajectory. The main view is also surrounded by buttons along

[2]Available at: `https://planner.muuto.com/`



■ **Figure 1.2** Screenshot of Muuto Product Planner tool with example configuration
Source: Muuto [14]

its edges. The navigation bar is positioned at the bottom across the entire application, while the company logo is displayed on the top left.

The tool follows a similar flow, starting with the selection of the starting point and then moving to the configurator process, which has open navigation. Components can be placed anywhere, unless their position is dependent on another component. Due to this flexibility and also the wide variety of products that it supports, the configurator is not restricted to generating configurations that are feasible to produce and can create impossible configurations.

The tool can display real dimensions and can reverse the performed changes using the undo and redo buttons. The configuration can also be saved on the server-side and later accessed using a unique code. Live price calculation is also performed, and there is a summary page, but it is not possible to order the configured product; instead, the user is redirected to a physical store locator.

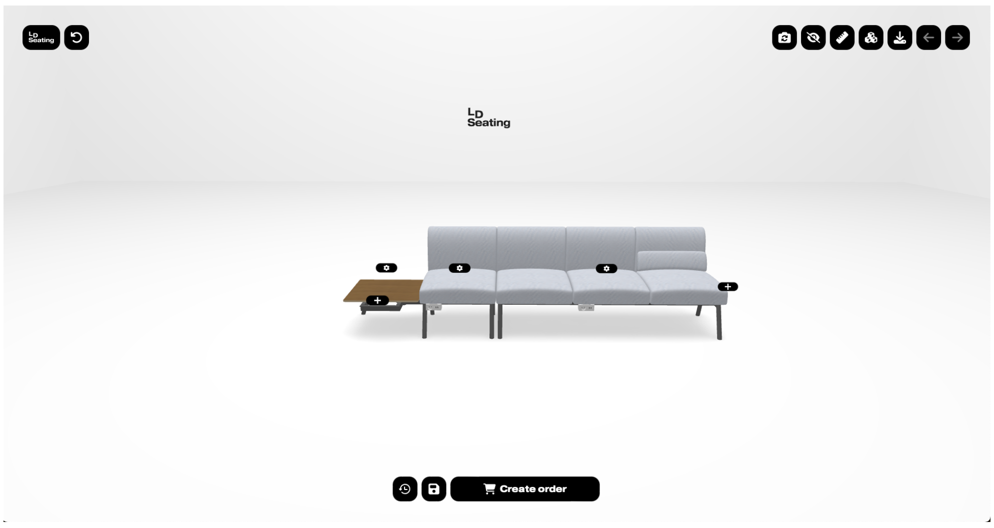
Furthermore, the designed configuration can be quickly shared with other users using email, or it can be downloaded in several file formats containing the 3D model itself. The application makes it possible to view the product in AR, directly in a web browser, albeit only on Apple devices using the Universal Scene Description Zip (USDZ) format and AR Quick Look. [15]

The application has multiple URL schemes that depend on the configuration phase, but they are not determined by the current product.

1.1.1.3 LD Seating Nido Configurator

LD Seating is company based in the Czech Republic that specializes in the production of chairs, armchairs, and sofas. [16]

The company uses a 3D web-based configurator to market the Nido modular seating system, which consists of elements that are designed to be combined in various ways.[3] [17]



■ **Figure 1.3** Screenshot of LD Seating Nido Configurator with example configuration **Source:** LD Seating [17]

The configurator consists of one large view, covering the whole application, which displays high-definition 3D models of the configured components. The buttons are placed around the entire view, on the top right, bottom center, and top left, where the company logo is also displayed. The controls for adding components and modifying properties are embedded directly in the 3D scene. When necessary, a panel opens to the right, allowing users to select components to add, adjust component properties, or change materials. The application is partially responsive, as the side panel opens fullscreen on smaller viewports; however, there are some issues with image and text overflows on mobile devices. The main view allows the user to observe the configuration from all angles along an orbital trajectory.

The flow of the application also includes selecting a starting point. The product configuration process itself uses open navigation. The tool assesses whether a component can fit into a space and, if not, prevents its placement, thereby restricting impossible configurations. At the completion of the configuration, a confirmation summary is presented; however, in this case, the price is not calculated live. The configurator cannot directly place an order for the product, but instead confirming results in the display of an inquiry form.

The resulting configuration can be downloaded as a file containing the 3D models. The configuration can also be saved server-side, which generates a unique link at which the configuration is accessible. The tool features

[3]Available at: `https://nido.ldseating.com/en/configurator`

a version history that stores each saved configuration for future access. Users can revisit these versions, while undo and redo buttons are also available. The resulting configuration can also be exported to a PDF file containing a list of components. The tool also has the ability to display real dimensions.

The application is a SPA, maintaining a consistent URL and changing it to reflect the location of a saved configuration (if one exists).

1.1.2 Available Toolkits

In this section of the analysis chapter, the focus shifts from specific 3D modular configurator applications to the fundamental toolkits that power the configurators. Although many configurators are bespoke and tailored to the specific needs of companies and their individual products, there are providers offering more generic and adaptable solutions. These offerings are highly relevant to this thesis, as the objective of this thesis is to create a product-agnostic toolkit, which means that there is a need to consider the way the configurator is set up by the business.

A variety of providers offer these toolkits for deploying product configurators, intending to provide semi-custom or fully custom solutions, as well as generic options. This section examines two particular toolkits to carry out a focused and relevant analysis. The choice of toolkits analyzed has been complicated by the fact that most providers are cautious about the details of the technology, typically revealing in-depth information only after a serious business inquiry. The choice was also based on factors such as the implementation approach and compatibility with modular products.
This section seeks to answer the following questions about the toolkits:

- **Administration**: How is the product configurator created and administered?
- **Assets**: How are assets stored and cataloged?
- **Product configuration**: How are the configuration options and rules defined?
- **Integration**: How is the tool integrated into other systems?
- **Pricing**: What is the cost of the offered solution?

1.1.2.1 Threekit

Threekit is a leading global company in visual commerce technologies that specializes in 3D product visualizations. The Threekit Platform, which enables clients to create interactive product experiences according to their needs, functions as an administrative application and has the capability to generate product configurators (see Figure A.1 in Appendix A). [18, 19]

The platform is very complex with many distinct features. At its core, it uses a catalog for storing all product data (the products themselves, materials used, configurable parts, etc.). The items in the catalog can then be loaded into The Treekit Player, which will display the models in 3D, with the option for users to change attributes (models and materials) that are tied to the item. The behavior of the configurator can be set up using item rules and logic that support conditions, queries, and even custom scripts. The platform also offers a data tables feature that is similar to spreadsheets and is designed to handle extensive configuration data and logic. The application also has a built-in asset editor for refining 2D and 3D assets and configuration options (see Figure A.2 in Appendix A). Models of the products can be uploaded in various 3D formats. The platform provides API integration with the leading e-commerce and ERP systems. [20]

The offered solution is still partially tailored to the client, which is the reason why the service does not have standardized pricing. Instead, the price is determined through a personalized quote. It is important to note that the analysis of Threekit presented here is based on the publicly available documentation of the platform. Direct access to the full suite of Threekit's tools is typically available only after formalizing a business agreement with the company.

1.1.2.2 Roomle

Roomle is an Austrian company focused on pioneering visual product configuration. They provide solutions for product visualization, room design, and product configuration. Roomle's solution, called Rubens, is described as "Open Full Logic 3D-Configurator". The tool utilizes both parametric and modular configuration mechanisms. The software allows integration with third parties through the use of an API. In addition, it supports integration with other front-end technologies on web and mobile platforms and has a built-in AR experience. [21]

A web application, Rubens Admin, is used to set up the configurator application. To add a product that can then be configured by customers, 3D models, and materials are uploaded to the admin application. Components are defined using RoomleScript language, which is loosely based on the JavaScript language. The design of the configurator itself can be tuned in the admin application as well. Multiple language variants can be defined for product names and descriptions. The configurator application (see Figure A.3 in Appendix A) runs on the client-side and can be simply embedded into a website. Additionally, a JavaScript library that can subscribe to events or modify the configurator can be utilized. A framework is also provided to utilize the configurator within an iOS application. [22]

In terms of pricing, Roomle's Rubens configurator with the listed capabilities is offered to businesses at a monthly fee of €1450. [23]

1.1.3 Summary of Existing Solutions

Features	IKEA Pax Planner	Muuto Product Planner	LD Seating Nido Configurator
Platform	Web	Web	Web
Navigation	Open	Open	Open
Visualization	Realistic	Realistic	Realistic
Placement options	Free	Free	Fixed points
Camera movement	Orbital	Orbital; Static	Orbital
Impossible configurations	No	Yes	No
Responsiveness	Yes	Yes	Yes
Price calculation	Yes	Yes	No
Purchase option	E-shop order	Store locator	Inquiry form
Save option	Server-side	Server-side; Local	Server-side; Local
Version history	Undo & redo	Undo & redo	Undo & redo; Multiple saves
VR or AR	No	Yes	No
Real dimensions	Yes	Yes	Yes

Table 1.1 Summary of key points discussed in the analysis of existing modular product configurators

All the product configurators analyzed have common characteristics, especially in terms of navigation and visualization styles, which remain consistent across different tools. Despite this, certain trade-offs were observed between them, especially regarding placement options. While some solutions offer users the freedom to position components anywhere, others restrict placement to fixed points. This variance stems from implementation complexity, as the fixed-point system is simpler, furthermore offering a better way to restrict impossible configurations. Another significant distinction was observed in the product finalization process, which ranged from the ability to place an order to being directed to a physical store.

The key points discussed in the analysis of modular product configurators are summarized in Table 1.1.

The user interface design of the analyzed configurations also displayed similarities, particularly in layout style, featuring a primary 3D preview on the left, a secondary view on the right, and buttons surrounding the primary view.

Analyzing the offered toolkit solutions proved challenging due to the information being closely guarded, as it is in the financial interest of the providers. However, the examined toolkits are very sophisticated solutions that are supported by large backend services, which are used for storing assets and facilitating the configurators functionality. The toolkits offer advanced features that allow for the definition of rules and logic, allowing companies to create configurators with a large amount of complexity. This indicates that these toolkits target a market composed mainly of larger corporations that require sophisticated solutions, which is also reflected in pricing.

1.2 Proposed Solution

This thesis aims to develop a new solution for the configuration of modular products. To do so, the proposed solution will incorporate the common characteristics identified in the analyzed solutions. The following paragraphs of this chapter should answer the important questions of who, what, why and how, detailing the key aspects of the proposed solution.

The main differentiation factor of this proposed toolkit is its emphasis on catering to small businesses. As the existing toolkits that were examined were costly and mainly aimed at larger companies, this solution aims to fill this market gap. To achieve this, it will be necessary to make some trade-offs ensuring the solution's adaptability and relevance across various product types without making the solution overly complex. Therefore, the proposed toolkit will prioritize simplicity and cost-effectiveness, following the best practices seen in larger-scale solutions but with a specific focus on the needs and capabilities of the target market.

The following chapter outlines the features that should be implemented in the solution proposed in this thesis.

The proposed toolkit is envisioned to be universal with regard to products, adaptable, and customizable, catering to a wide variety of modular products and industries. The solution should be simple for businesses to deploy and manage, without the need for extensive technical resources, ensuring that it is straightforward for smaller businesses to maintain and operate effectively.

1.2.1 Requirement Engineering

Following the overview of objectives and the definition of the target market, it is necessary to formulate precise requirements for this solution. Detailing these features and characteristics is crucial for successful implementation. Thus, the description of requirement engineering for this solution will be provided here.

The process of requirement engineering for software products involves gathering, analyzing, selecting, and managing requirements. It focuses on interpreting and understanding the goals, needs, and beliefs of stakeholders and transforming them into specific requirements. [24]

There are many ways to categorize software requirements, such as audience-oriented categorization or using the FURPS method, which classifies requirements based on functionality, usability, reliability, performance, or supportability. [25]

Given that the majority of requirements for the solution fall either into the functionality or usability category, the requirements in this section are separated only into the following common [24] two main categories:

1. Functional requirements: These requirements describe what the system should be able to do. They specifically outline the system's behavior and its interactions in specific situations.

2. Non-functional requirements: These requirements put constraints on the solution that meets the functional requirements, rather than being focused on specific behaviors of the system. They are often, among others, focused on performance, security, accessibility, and compatibility.

To manage and prioritize these requirements, each is assigned an approximate priority level using the MoSCoW method. This approach classifies the requirements into four distinct categories:

1. Must: Requirements crucial for the final solution.

2. Should: Requirements to be implemented if feasible.

3. Could: Requirements that are desirable but not essential.

4. Won't: Nice-to-have requirements that most likely will not be implemented in this solution.

This method helps to plan and allocate resources throughout the implementation phase. [25]

Additionally, each requirement is given a rough estimate of the implementation difficulty, separated into three categories:

1. Simple: Requirements that are straightforward to implement and require little time and few resources.

2. Intermediate: Requirements that pose moderate challenges and demand a considerable amount of resources, time, and problem-solving.

3. Complex: Requirements that are highly challenging and involve substantial resources, time, and expertise.

This preliminary assessment aims to classify the requirements without relying on specific rigid criteria for each category. This method is specifically used only for functional requirements, as non-functional requirements affect the software's functionality and user experience through abstract constraints, making them unsuitable for the same difficulty estimation approach.

1.2.1.1 Functional Requirements

F1: 3D product visualization

The tool shall offer users 3D visualization of their configured product, employing realistic models to accurately represent the components used and their characteristics.

- **Priority:** Must
- **Difficulty:** Complex

F2: Dynamic orbital camera controls

The tool should have dynamic orbital camera controls that allow users to view the product in the 3D product visualization (see requirement F1) from any angle by rotating, panning, and zooming the camera around the product. This feature aims to provide an engaging visual experience that allows users to examine the product with a 360-degree view. The controls should be intuitive, allowing for seamless navigation through mouse actions or touch gestures depending on the device used.

- **Priority:** Must
- **Difficulty:** Simple

F3: Modularity configuration mechanism

The toolkit should incorporate modularity mechanisms that allow users to configure products by adding, removing, or modifying components within the overall product or in relation to other components. Different modules may be available for each component, and the toolkit administrator should have the ability to designate them either as optional or mandatory, thereby enhancing the flexibility of configuration.

- **Priority:** Must
- **Difficulty:** Intermediate

F4: Component interactivity

The configurator should support interactivity with each component of the product. Users should be able to select components directly within the 3D visualization (see requirement F1), which should allow them to change attributes of the components, remove them, or swap them with alternative options (see requirement F3). The changes made by the users should be immediately visible, allowing for an iterative and engaging customization process. Moreover, the components that are interacted with need to offer feedback, such as highlighting, in order to assist users in navigating the accessible customization choices.

- **Priority:** Should
- **Difficulty:** Complex

F5: Open navigation

The configurator should offer high flexibility in the order of configuring components and attributes, avoiding a linear step-by-step configuration and enabling all changes to be performed at any point during the configuration process. This flexibility enhances the user's ability to navigate freely among various different components of the product (see requirement F3).

- **Priority:** Should
- **Difficulty:** Simple

F6: Fixed point component placement

In alignment with the modularity configuration mechanism (see requirement F3), the configurator should enable components to be attached to other components or the whole product at predefined fixed points. While this approach restricts the potential solution space, it greatly streamlines the configuration process from the user side and helps to ensure that the configured product remains within the realm of feasible configurations.

- **Priority:** Should
- **Difficulty:** Intermediate

F7: Component collision detection

The tool should incorporate a collision detection system to prevent components from being positioned in such a way that would result in physical overlaps during configuration. This feature is essential to maintain the realism and feasibility of the configured product.

- **Priority:** Should
- **Difficulty:** Complex

F8: Material color configuration

Users should be able to modify the appearance of materials of components and products through a selection from a palette of colors. The chosen appearance should immediately be reflected in the 3D visualization (see requirement F1) of the configuration.

- **Priority:** Should
- **Difficulty:** Complex

F9: Configuration review

Before the configuration process is finalized, users should be presented with a review page that allows for a detailed examination of their product configuration. This feature should provide a summary listing all selected components and any other parameters. In addition, users should be able to return to previous configuration steps to make any necessary adjustments.

- **Priority:** Should
- **Difficulty:** Simple

F10: Configuration processing

At the end of the configuration process, users should be optionally presented with a confirmation button, provided that a specific confirmation action has been set up for the product. This button is intended for users to confirm their choices and trigger a predetermined action, such as calling a webhook or being directed to another page, as specified by the administrator. This should allow for a smooth transition, where, upon configuration confirmation, the user is engaged in a follow-up action, like a checkout process or being guided to a physical store locator page. The ability to perform a custom API call at the end of the configuration process provides a flexible way to integrate the configurator with different systems or processes, thus improving its functionality and delivering a seamless user experience from start to finish.

- **Priority:** Should
- **Difficulty:** Simple

F11: Inquiry form

As an extension of configuration processing (see requirement F10), the administrator should be able to set the product's confirmation action to trigger an inquiry form. In this scenario, when users click on the confirmation button, they should encounter a form asking for their contact details. Once completed, the created product configuration along with the user's contact information should be sent to an API predefined by the toolkit's administrator. This allows for a standard inquiry form process directly within the configurator application.

- **Priority:** Should
- **Difficulty:** Simple

F12: Configuration saving and retrieval

The tool should allow users to save the current product configuration, allowing them to pause the customization process without losing progress. Users should have the ability to easily access and resume editing their saved configurations at a later time.

- **Priority:** Could
- **Difficulty:** Intermediate

F13: Undo and redo actions

The configurator should integrate undo and redo functionality, enabling users to easily revert or reapply changes made anytime during the configuration process.

- **Priority:** Should
- **Difficulty:** Intermediate

F14: Interface appearance customization

The interface of the configurator should offer customizable options, enabling the toolkit's administrator to tailor the style, color scheme, and images to align with the branding and design of the business employing the toolkit.

- **Priority:** Could
- **Difficulty:** Simple

F15: Interface texts customization

The toolkit should provide a way for the administrator to change the textual contents of the configurator's interface, ensuring that the language, tone, and terminology used are perfectly aligned with the business's needs and reflect the business's terminology and branding.

- **Priority:** Could
- **Difficulty:** Intermediate

F16: Visual catalog management

The toolkit should provide administrators with the ability to visually manage the catalog of configurable products and their components. The visual preview of the components provided within catalog management should mirror the 3D previews in the actual configuration process (see requirement F1). This management system should allow administrators to add, update, or remove products and components, along with specifying their precise mounting locations (see requirement F6), directly through a visual interface.

- **Priority:** Must
- **Difficulty:** Complex

F17: Product properties and attributes management

The toolkit should provide a way to manage the properties and attributes of the products and components in the catalog (see requirement F16), allowing administrators to define and adjust the characteristics that users can configure. It should allow for the detailed specification of each component's features, such as color options, material types (see requirement F8), and any other attributes that define it.

- **Priority:** Must
- **Difficulty:** Complex

F18: Real-time price calculation

If the configured attributes and components have prices predefined by the toolkit's administrator, the configurator should automatically update and display the price of the whole customized product with every change made. The tool should be capable of dealing with different currencies.

- **Priority:** Could
- **Difficulty:** Intermediate

F19: AR viewing capabilities

The configurator should extend its visualization features (see requirement F1) to include AR viewing capabilities, enabling users to project their configured products into their real-world environment through their device's camera. In case the device they are using does not have AR capability, the tool should provide a seamless way for the user to open the configuration in AR on another device that does have such capability.

- **Priority:** Won't
- **Difficulty:** Complex

F20: Parametric configuration mechanism

As a complement of the modularity mechanism (see requirement F3) the toolkit should incorporate parametric mechanisms that allow users to configure products by setting parametric values on the configured components when interacting with them (see requirement F4). The toolkit administrator should have the ability to create configurable parameters on the modular components, along with the types and possible ranges of values, enlarging the solution space of configuration.

- **Priority:** Won't
- **Difficulty:** Complex

F21: Real dimensions visualization

The tool should provide users with a visualization of the real dimensions of the configured products. The visualization should accompany the configured product in the 3D view (see requirement F1) and should provide realistic measurements of dimensions in actual, real-life units.

- **Priority:** Could
- **Difficulty:** Intermediate

1.2.1.2 Non-Functional Requirements

NF1: Multiplatform compatibility

The solution should work smoothly on various operating systems and devices, such as desktop and mobile platforms. This ensures that the solution is accessible to a wide audience, regardless of their preferred technology, thereby maximizing user engagement and reach.

- **Priority:** Must

NF2: Responsiveness

The user interface should be responsive, adapting to viewport sizes and resolutions on different screens, ensuring an optimal viewing and interaction experience across all supported devices.

- **Priority:** Must

NF3: Self-hostable architecture

The toolkit should be designed with the intent of being deployed and hosted on a business's preferred infrastructure, whether on-premises or in a private cloud. This facilitates greater control over the data and security according to the operator's policy, as well as flexibility for possible modifications.

- **Priority:** Must

NF4: Infrastructure needs

The toolkit should ideally operate with lightweight infrastructure needs, possibly leveraging the resources that may already be used to offer the products. The configurator application is expected to operate primarily on the client side, requiring only minimal back-end support, possibly making use of a simple serverless architecture if needed. This approach reduces the demand for maintenance and is for this solution cost-effective, improving existing operations without necessitating significant new investments in infrastructure.

- **Priority:** Must

NF5: Maintainability

The codebase and architecture should be designed to facilitate easy maintenance, straightforward updates, modifications, and enhancements. To ensure that the toolkit remains robust and flexible for future needs, industry standards and best practices should be adhered to during implementation. The tool should require minimal routine maintenance by the administrator. The quality of the codebase must be maintained to a high standard through the use of rigorous testing.

- **Priority:** Must

NF6: Documentation

To support maintainability (see requirement NF5) and ease of use, comprehensive documentation is essential. This should cover the configurator's setup, deployment, customization options, managing products, components, and also possible user interactions. Providing comprehensive and detailed documentation guarantees that administrators and developers can efficiently employ and customize the configurator to suit their individual requirements. It also serves as a valuable resource for troubleshooting, further development, and maximizing the potential of the tool.

- **Priority:** Should

NF7: Performance

The toolkit should ensure optimal performance under typical usage load, with swift loading and quick response times across all compatible devices, particularly those with lower processing power. Therefore, the application should aim to achieve performance of more than 30 frames per second on average consumer computers or mobile devices.

- **Priority:** Must

NF8: Multilingual support

The configurator should offer multilingual support. Administrators should be able to simply add, remove, or update languages, thus making it easier to adapt the interface for different language versions. This capability builds on the interface text customization requirement (see requirement F15), extending its scope to include different language options. Users should be provided with a simple method to select their preferred language.

- **Priority:** Could

Chapter 2

Design

Selection of used technologies, establishment of the domain model, and initial design of the user interface layout.

2.1 Technologies

Choosing the appropriate technologies is crucial and will have a significant impact on the overall effectiveness and excellence of the developed solution. Selecting technologies requires assessing different technological choices based on all the factors that will enable them to meet the requirements outlined in the preceding chapter. The right technology stack can also decrease the time spent on development, minimize costs, and ensure the solution remains relevant in the future.

2.1.1 Platform

In considering the foundation for the 3D modular product configurator, given the multiplatform requirement (NF1), two distinct development approaches were evaluated: applications specifically designed for desktop and mobile platforms or a web application.

Desktop and mobile applications can provide a better overall experience tailored to the specific platform and potentially be more performant as they can utilize the hardware better; however, in this case, they come with serious drawbacks. Having multiple applications that are designed for different devices would increase the amount of maintenance and development work required because each version would need to be managed (at least in some capacity) separately. Furthermore, accessibility for users would be dramatically diminished since they would need to download and install the application prior to using it and subsequently manage any updates that may arise.

Developing separate desktop and mobile applications presents such significant challenges that the disadvantages far outweigh the advantages; therefore, a web application was selected for its better alignment with the project's requirements (this is also consistent with the norm in this space, as the majority of existing solutions that were analyzed in the previous chapter are web-based). There are also several key factors in favor of this solution: it can be accessed from almost any device with an internet connection and web browser, it is cost-effective as it possibly utilizes existing website infrastructure, and it has streamlined maintenance needs. The application will be focused on the front-end, as that is where the configuration process will be happening.

Choosing a front-end-only architecture offers several advantages that align closely with the objective of the project. The nature of modular product offerings usually does not demand frequent updates, which makes serving static content feasible and efficient. While there are some aspects of the solution's requirements that might require back-end interaction, such as more complex final processing of configuration or saving of configurations, these can, however, be in some way be handled through external API calls or a lightweight serverless architecture. These aspects do not represent the highest priority requirements, and the core functionality can be accomplished using just front-end technologies.

This architectural choice significantly simplifies the development process. Since the application can be hosted on an existing web server architecture, deployment is also simplified, thus reducing the costs. This architecture is particularly advantageous for businesses that already have web hosting, such as those operating an inquiry form, but wish to enhance their configurable product offerings presentation without additionally complicating their infrastructure. Compared to a full stack solution, maintenance is reduced and, from a security point of view, the attack surface is greatly minimized.

2.1.2 3D Visualization Technology

To fulfill the requirement of 3D visualization (F1), a library that will allow 3D graphics to be rendered in the browser will have to be used. However, the range of options for this particular technology is quite restricted.

Historically, the integration of 3D graphics required the use of external plugins, primarily Adobe Flash Player. The evolution of web standards, particularly the introduction of HTML5, has revolutionized this aspect, and it is now possible to render 3D graphics directly in the browser, eliminating the necessity for any plugin. [26]

Web Graphics Library (WebGL) is a standard 3D graphics API for web browsers. It is based on OpenGL ES and can be used inside the HTML canvas element. WebGL is supported in all major desktop and mobile browsers.[1] It is utilized using C-like shading language (OpenGL Shading Language) and JavaScript. [27]

Currently, there are no significant alternatives to WebGL. WebGPU aims

[1]WebGL browser support details: `https://caniuse.com/webgl`

to be a successor to WebGL; however, as of February 2024, it is in a state of ongoing development and has not yet been finalized or supported in web browsers.[2] [28]

2.1.2.1 WebGL Framework

Direct WebGL programming is very powerful and offers fine-grain control, necessitating extensive code to be written in both JavaScript and its shader language. Fortunately, there are several frameworks built on top of WebGL that provide high-level abstractions and access. These frameworks can significantly reduce the amount of code required to achieve what would otherwise take hundreds of lines when using bare WebGL, often condensing it into just a few lines. [26]

Three.js was selected from a range of frameworks, including Babylon.js [29], that are designed to streamline the process of developing in WebGL. This decision was made after considering several important factors. Three.js is considered an undisputed leader in this category, having the biggest community support, which can be evidenced by its popularity and the volume of contributions on GitHub.[3] It is also open source, published under the MIT license, offering great freedom in development and distribution. Furthermore, Three.js uses the best practices of 3D graphics; it is lightweight, easy to use, cross-platform, and contains many prebuilt assets. [30]

2.1.3 Front-End Framework

Leveraging front-end frameworks significantly enhances the development of web applications by addressing common front-end challenges. These frameworks often provide a structured approach for creating maintainable and reusable components, optimizing data manipulations, employing common design patterns, and ensuring that the user interface remains in sync with the underlying state. Various frameworks and libraries are available, such as React, Vue.js, or Angular, each with different benefits and drawbacks. The choice of which framework to use often involves complex decision making, influenced by specific project needs, team skills, and the unique characteristics of each framework. [31, 32]

For this solution, the decision has been greatly influenced by the selection of WebGL framework, which is Three.js (see previous section). That is because of the React Three Fiber (R3F) library, which offers a seamless integration of Three.js into the React ecosystem. R3F is a React renderer, enabling the direct use of Three.js components as React components. The integration is optimized, with the Three.js components rendered outside React's rendering process,

[2]WebGPU browser support details: `https://caniuse.com/webgpu`
[3]Three.js GitHub: `https://github.com/mrdoob/three.js`

therefore, having minimal overhead. Moreover, it is comprehensive, meaning that all Three.js features are exposed and accessible using this library. [33]

In addition, the Drei library, built on top of R3F, introduces a collection of useful components, abstractions, and helpers. These additions streamline the development with Three.js and React even more. [34]

These libraries make React an attractive choice for this project. To see how the code differs when aiming to achieve similar objectives, refer to Code listing B.1 in Appendix B for the plain Three.js version and Code listing B.2 in Appendix B for the R3F version, both creating a simple 3D red cube.

React is a user interface library created at Facebook in 2011, but soon after became open source. React has gained widespread acclaim across many projects and has been continually developed since its inception. It emphasizes component-based architecture, where reactive components are written in JavaScript (or TypeScript) combined with HTML-like markup code, facilitating the creation of dynamic user interfaces. [35]

React itself is just a user interface library that lacks more sophisticated functionalities, such as routing. There are several frameworks compatible with React, such as Next.js or Gatsby.js, which offer advanced features like caching, routing, server-side rendering, search engine optimization, and more [36]. However, because the dynamic content of this web application is highly influenced by user interactions, the solution would not benefit from these frameworks. Therefore, the decision was made to maintain simplicity, opting for the utilization of select libraries for advanced features rather than complex frameworks.

Prioritizing speed, simplicity, and minimal configuration requirements, Vite.js has been chosen as the build tool and development server. [37]

2.1.3.1 CSS Framework

The development of a product configurator requires custom components. To define the styles of these custom designs, it will be necessary to utilize Cascading Style Sheets (CSS).

TailwindCSS is a utilitfy-first CSS framework. It enables the creation of custom designs using predefined CSS utility classes, directly applicable in the React markup language, eliminating the necessity of directly writing CSS. It is highly customizable, has comprehensive and illustrative documentation, and makes it easy to create responsive designs. The framework allows for a fast development process, however, it needs to be integrated carefully, as the direct combination of style classes with the rest of the code of the component can make the codebase look very disorganized. [38]

It was chosen for this project for its ability to accelerate the development process and to help meet the responsiveness requirement (NF2).

2.1.4 Programming Languages

The selection of programming language is predetermined by the already chosen technologies and libraries, necessitating the use of React markup and JavaScript in some form.

Fortunately, with Vite.js's ability to transpile TypeScript to JavaScript, and given that type declarations are exported from the chosen JavaScript libraries, TypeScript can also be used. [37]

TypeScript is a programming language created by Microsoft that extends JavaScript by implementing strong typing. Strong typing helps detect bugs during development, reduce errors, and improve overall code quality. It also allows for tighter integration with code editors, enabling features such as autocompletion or inline documentation. All code written in TypeScript is transpilable to JavaScript, which means that it is compatible with existing libraries and frameworks. [39]

Given these advantages, TypeScript will be used in this project in place of JavaScript, ensuring a maintainable, high-quality codebase.

2.1.5 Additional Libraries

To enhance the functionality in a way that the frameworks described above do not support natively, several additional libraries will be used in the solution.

2.1.5.1 Routing

To improve the application's user experience with navigable URLs, allowing redirection, linking, or bookmarking pages, the use of a routing library is essential, as in a SPA, all content is served on a single address by default.

For React, the leading library for routing is React-Router, which will be utilized for this purpose in this solution. This library has been chosen for its widespread use and robustness. Its use promises that the application supports dynamic and user-friendly navigation. [40]

2.1.5.2 Language Support

To address the requirement of user interface text customization (F15) and multilingual support (NF8), an internationalization and localization library is essential. Such library enables the dynamic sourcing of user-interface texts from separate files according to the application's current settings. Consequently, on the basis of the extensive set of features and extensions offered, the i18next internationalization framework was chosen for this project. [41]

2.1.5.3 State Management

State management is a critical element of React applications that links the internal state directly to the user interface. Although React offers a basic mechanism by default, complex applications highly profit from a sophisticated state management library that manages state updates and interface redraws in a comprehensive manner. [42]

Although there are numerous different state management libraries, each with its advantages and disadvantages (such as Zustand, Redux, MobX, etc.), for this project, Valtio has been chosen. Valtio stands out for its extremely minimalistic API, while being very flexible with data structures. It uses proxies and allows for direct mutations, making the handling of state as intuitive as working with regular JavaScript (or TypeScript) objects. [43]

These characteristics make Valtio highly beneficial for this application, as it will simplify mapping the underlying state to 3D objects, which will be necessary to fulfill the 3D preview requirement (F1).

2.1.5.4 Validation

To guarantee the integrity and structure of the loaded data with values that adhere to specified constraints, validation is crucial. Validation libraries have been designed for this purpose, and in this project, Zod has been chosen as the data validation library.

Zod allows the inference of TypeScript types directly from the created data schemas, meaning a single schema definition can be used for parsing as well as type checking. Furthermore, Zod is also very lightweight, and its powerful yet straightforward API for schema definition makes it a compelling choice for this task. [44]

2.1.6 Development Tooling

The effectiveness and resilience of the development process depends on the selection of development tools. These tools not only simplify code creation, but also help ensure code quality, version control, and improve efficiency in collaborative work.

2.1.6.1 Version Control

Version control plays a crucial role in modern software development, allowing developers to track changes to source code over time. For this project, Git has been chosen as the version control system due to its widespread adoption and robust feature set. Git is a distributed system that allows for easy collaboration with other developers, as well as the creation of independent adaptations of the codebase through forking, all while preserving a connection to the original repository for future updates or integrations. In this way, a unique version of the application codebase can be tailored, in case it is required to meet specific custom business needs. [45]

GitLab is a DevOps platform that is used to host Git repositories, offering a wide range of features. The primary benefit for this project lies in its CI/CD pipelines, which can be used for automated testing, linting, and building of the project. This automation speeds up the development process. [46]

Conventional Commits is a convention for naming Git commit messages in a descriptive form, creating rules that communicate the scope of changes and allow the creation of further automations on top of them. This convention will be used to ensure order and clarity in the Git repository. [47]

To enforce the use of conventions in commits, Husky, a tool for utilizing Git hooks, will be integrated into the development workflow. Husky allows for the setup of actions that are triggered at specific points in the Git lifecycle. This way, the project can automatically lint commit messages to ensure that they follow the Conventional Commits format, as well as check contents of the commits. [48]

2.1.6.2 Package Manager

To streamline the process of managing the dependencies of the project, npm has been selected as the package manager due to its wide adoption and compatibility within the ecosystem. [49]

2.1.6.3 Formatters

For ensuring code consistency and style guidelines adherence, especially when working with TypeScript, due to the nature of the language, linters and code formatters need to be used.

ESLint will be used as a linter to help identify and fix problems in TypeScript code, removing problematic patterns, promoting best practices and code consistency. [50]

Prettier will automatically format the code to meet style guidelines, making it easy to read and reducing formatting discrepancies. [51]

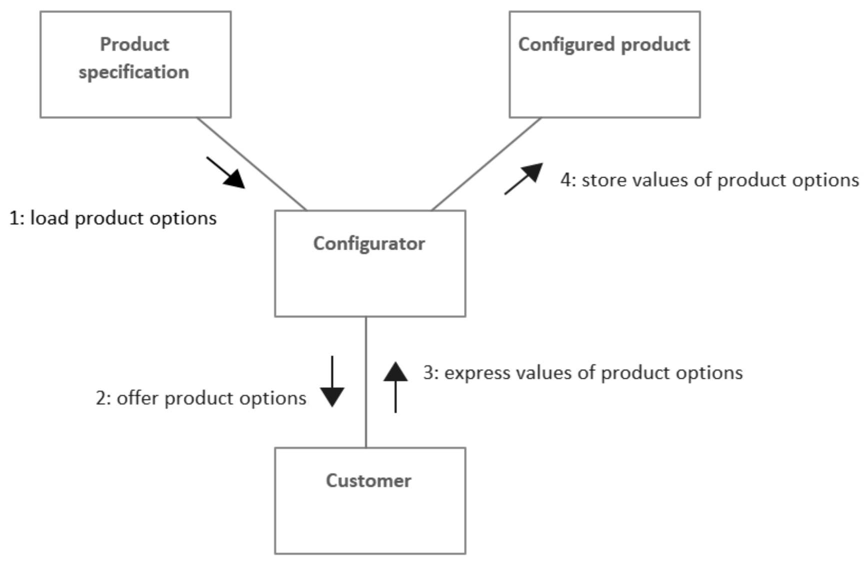
2.2 Domain Model

Before implementation, an important part of the design process is the description of a domain model. This outlines the concepts the tool will work with and defines the interactions of entities, establishing an important context that serves as a foundation for building the toolkit.

The domain of the solution is based on the basic functionality of product configurators [52], which is illustrated in Figure 2.1. At the core of the tool are two actors: the customer and the configurator, as well as two models: the product specification, which defines the possible configuration options, and the configured product, which stores the options the customer has chosen. The flow is then as follows:

1. The product specification, which stores the product options, is loaded into the configurator.
2. The configurator presents the customer with the product options.
3. The customer expresses their desired values of the product options to the configurator.
4. The configurator stores the desired values of the product options as a configured product.

The domain model for this solution is based on these concepts; however, it is more detailed, as more complex relationships and objects are necessary.



■ **Figure 2.1** Diagram of core functionality of product configurators
Source: Adapted from [52]

Unified Modeling Language (UML) diagram can easily represent the domain model. [53]

The UML diagram that illustrates the domain model of the entities related to the proposed toolkit is presented in Figure 2.2. This model serves as a blueprint for the architecture of the system, outlining a coherent structure that aligns with the project objectives described in the previous chapters.

The domain model presents a structure of product specifications contained in a catalog, the dependencies between different entities, and illustrates the progression from generic specifications to concrete user configurations. This model aims to offer a clear insight into the design rationale behind the tool.

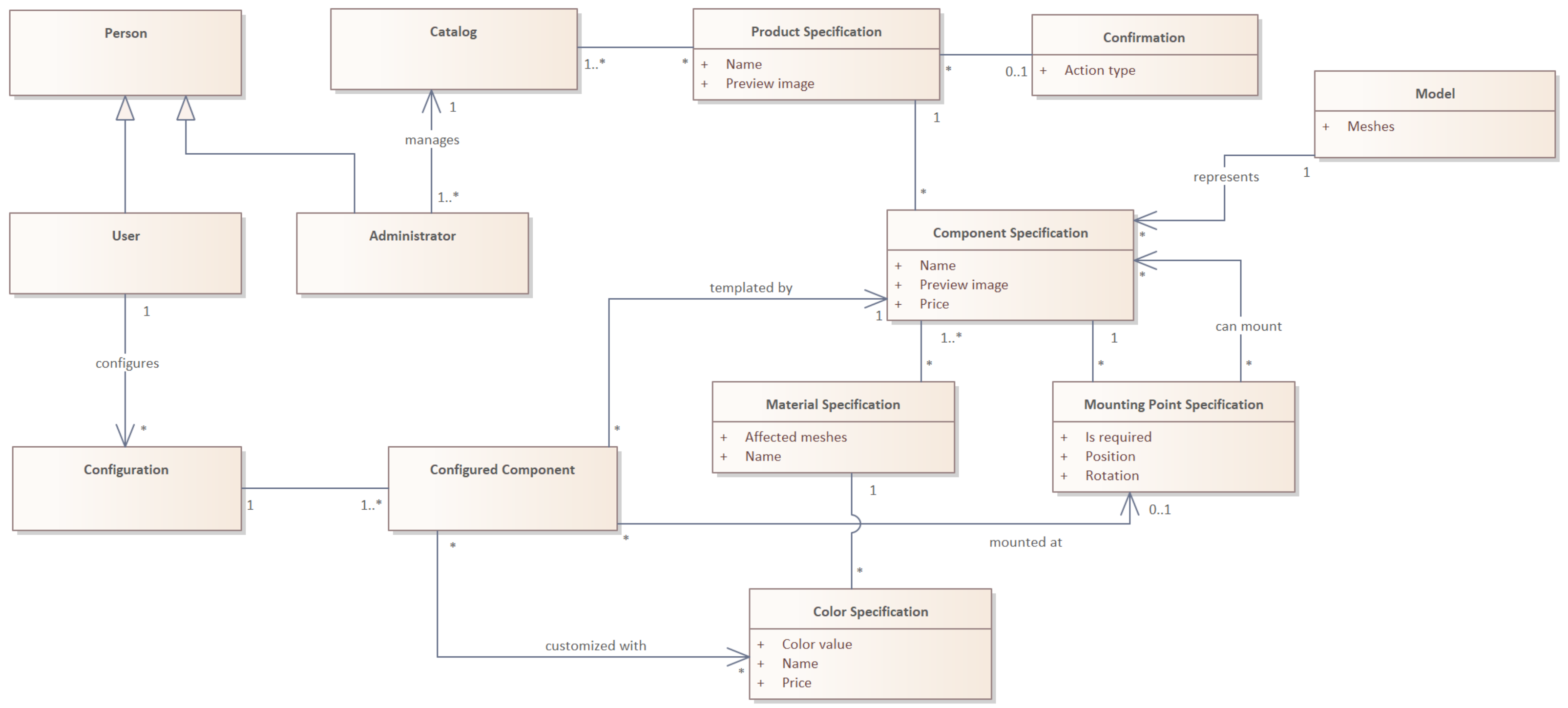


Figure 2.2 Domain model as UML diagram

2.2.1 Catalog

The catalog forms the backbone of the proposed toolkit and defines the blueprint for customizable products. It encompasses a variety of product specifications, each defining a configurable product along with all its potential customizations. This section looks at how these specifications lay the foundation for user-driven product configuration. The entities from the diagram in Figure 2.2 discussed in this section are as follows:

- Catalog
- Product Specification
- Component Specification
- Mounting Point Specification
- Material Specification
- Color Specification
- Model
- Confirmation
- Administrator

Product specification acts as an overreaching comprehensive concept that encompasses all aspects of a given product. The entity may be associated with confirmation, an action to finalize the configuration of the product, fulfilling the need for processing of the configuration (see requirement F10). Given that the tool focuses on the handling of modular products (see requirement F3), it is imperative that each product consists of various components. Therefore, the product specification is made up of component specifications, which represent all the various possible components that the product can have.

To meet the requirement of 3D visualization (see requirement F1), component specifications must be linked to a model featuring 3D meshes. This model acts as a representation of the component that will be presented during the configuration process. In addition to this, the component specification is composed of material specifications and mounting point specifications.

The material specification is needed with respect to the requirement of material color configuration (see requirement F8). It describes the materials of a given component that the user can customize, with each material specification specifying which part of the component this material influences, thereby enabling preview updates as the user makes selections. In addition, the material specification consists of color specifications that define the possible colors that this material can take on in the configuration process.

Mounting point specifications are introduced to address the requirement of fixed point component placement (see requirement F6). They represent points on a component to which other modular components can be attached. The mounting point specification includes the relative position and rotation of the point on the component specification, the requirement for a component's presence at this point, and possible component specifications that can be mounted on the point.

All these specifications are maintained in the catalog by the administrator, who has the authority to modify any properties. The tool then utilizes these specifications to enable the configuration of customized products.

2.2.2 Configuration

Transitioning from specifications to actual configurations, the configuration section describes how users bring customizable products to life through the toolkit. It illustrates how configured components are the building blocks of user-generated configurations, realizing the transformation from a generic template into a product uniquely tailored to individual preferences. The entities from the diagram in Figure 2.2 covered in this section include:

- Configuration
- Configured Component
- User

Users of the application create configurations that represent configured products. The cornerstone of the configuration lies in its configured components. The specifications described in the previous section serve as templates for these configured components. While the specifications outline all the configuration options, the configured component indicates a particular set of options selected by the user. Beyond storing the component specification the configured component customizes, it also stores the mounting point specification to which it is attached, as well as the selected color specifications of its material specifications. Thus, a configuration is composed entirely of these individual configured components.

2.3 User Interface

The design of the user interface plays an essential part in the development of such a tool because it significantly influences user satisfaction when interacting with the tool. Good preparation of user interface design helps to determine the direction and streamline the implementation, as well as making clear from the beginning how to deal with factors such as responsiveness.

In this section, low-fidelity wireframes are used to depict the proposed user interface, highlighting the layout and architecture of the application.

This approach captures the most important information at this stage, leaving the finer details to be refined as part of the implementation phase.

An article [54] used The Configurator Database Project to analyze common design elements of product configurators. The study identified several key design elements that were prevalent in the majority of configurators analyzed. The following key insights of commonly used designs from the article are relevant to this thesis:

- At the end of the configuration process, a summary of selected options is presented.
- To present the products that can be configured, detailed images are used.
- Horizontally positioned navigation bar is always visible.
- Order button is clearly visible and available for completion purposes.
- Prices of the components are accessible in all phases of the configuration.
- Logo of the business is prominently displayed.
- Users can navigate within their configurated product.

Furthemore, design guidelines from the Website Tools and Application with Flash report by Nielsen Norman Group [55] were also considered. Although the report analyzed and offered guidelines for the design of Flash applications, it is still relevant to this thesis as many of the applications were 3D-based or featured product configuration. In particular, the following key principles were followed:

- Core task must at center of the application.
- Complex features that detract from the core task overwhelm users and increase learning time and should be avoided.
- The interfaces must not be overcrowded and should prioritize presenting the most important information.
- The interactions should be consistent within similar components of the application.
- The design should respect and reuse good design principles of similar solutions that users may already feel familiar with.
- Technical details of the solution should be hidden from the user.
- Feedback to user's actions should be imminent.
- If available, prices should always be displayed.

- Visalization should match the input of users.
- Popup windows should include close buttons.

The design is therefore based on the described insights and respects the intuitive design principles of similar solutions that users may already be familiar with, ensuring an intuitive and efficient interface for users.

The configurator as an application is specific in that it primarily centers around the configuration process, with this interface being the most important and all other interfaces being secondary. In this section, the design of this configuration screen will be introduced, as well as the introduction and confirmation screens.

The common element of all screens is a top bar with the logo of the business that operates the configurator, which should redirect the user to the main website of the company when clicked. All of this should be customizable in the admin settings of the toolkit.

2.3.1 Configuration Screen

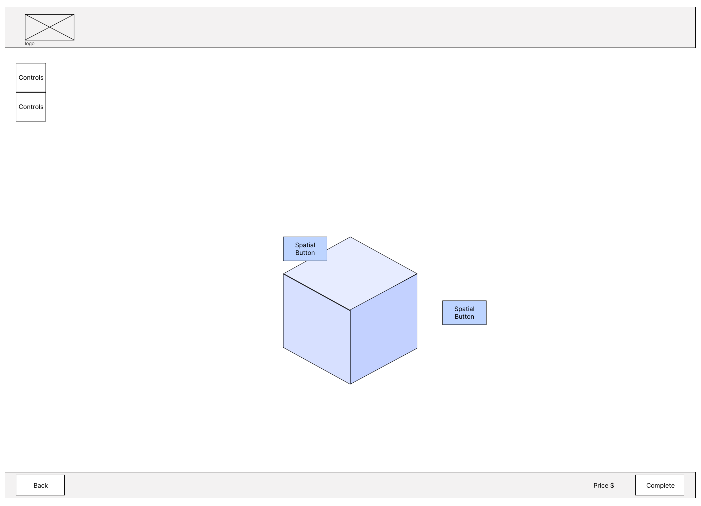
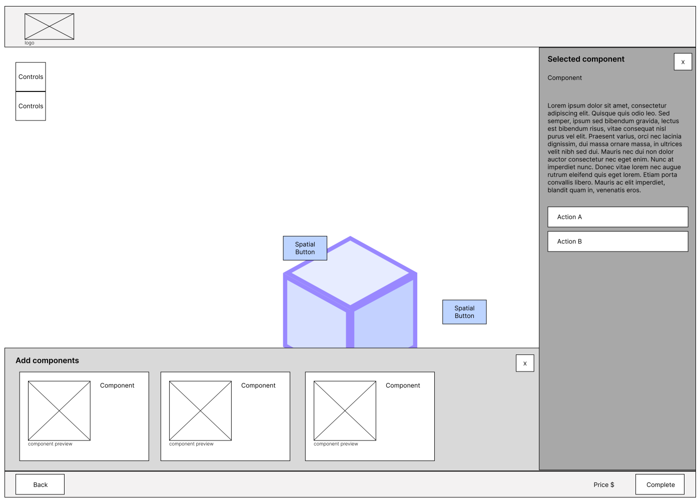


Figure 2.3 Wireframe of configuration screen

The configuration screen is presented to the user during the configuration process. The screen is dominated by the 3D preview of the configured product, featuring interactable components and spatial buttons for the addition of components into the configuration. This design follows the requirement of open navigation (see requirement F5). Control buttons are placed in the upper left corner within the 3D preview, symbolizing their direct relation to the preview, yet maintaining their distinctiveness as a separate element. At the bottom of the screen, there is another bar, this one containing buttons that allow users to go back or to finalize the configuration. The wireframe of the configuration screen in its default state is shown in Figure 2.3, where the 3D preview of the product is represented by a blueish cube.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque quis odio leo. Sed semper, ipsum sed bibendum gravida, lectus est bibendum risus, vitae consequat nisl purus vel elit. Praesent varius, orci nec lacinia dignissim, dui massa ornare massa, in ultrices velit nibh sed dui. Mauris nec dui non dolor auctor consectetur nec eget enim. Nunc at imperdiet nunc. Donec vitae lorem nec augue rutrum eleifend quis eget lorem. Etiam porta convallis libero. Mauris ac elit imperdiet, blandit quam in, venenatis eros.

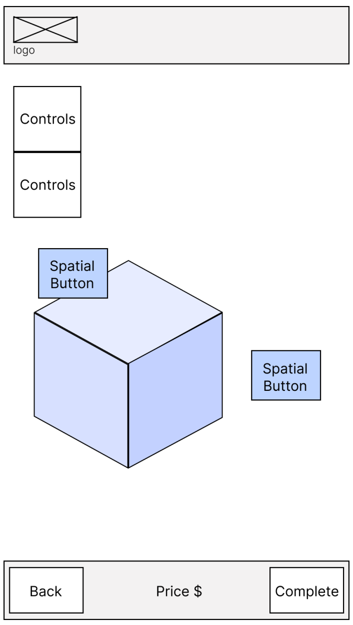
■ **Figure 2.4** Wireframe of configuration screen with panels

This default view, as outlined in the previous paragraph, maximizes the viewport with the most important presentation, which is the 3D preview of the product. However, at some stages of the configuration process, it is also necessary to present the user with further information. Therefore, upon selecting a component within the 3D preview, the component should become highlighted and, following the approach of existing solutions analyzed, a side panel with detailed information about the selected component will emerge from the right. If necessary, another panel with further options presented to the user may appear at the bottom. This design strategy maximizes the screen space for the important elements while still being flexible enough to present additional information in a streamlined way. The wireframe of the interface with panels that contain additional information and options presented is shown in Figure 2.4.

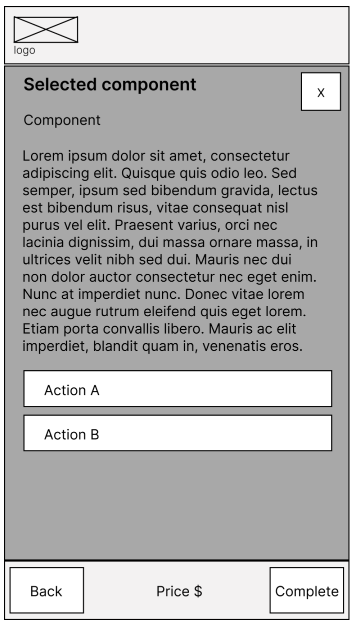
To ensure responsiveness, a mobile version of the interface should also be outlined. The prioritization of the 3D preview in the default state makes this simple, as this means that on smaller viewports the elements are presented in the same way, with the preview being in different aspect ratio, which is easily compensated for by the 3D preview taking on different zoom level. The outline of this wireframe is presented in Figure 2.5.

The wireframe for the mobile version of the interface with the presented side panel is shown in Figure 2.6. At this reduced viewport size, the side panel can maintain the same internal layout but to be usable it needs to occupy the whole 3D preview. This will need to be kept in mind when utilizing interactions with the 3D preview, as it may not always be fully visible when the panels are active.

The design remains mostly consistent on both small and large viewports while still being adaptive and responsive. This ensures that the familiarity with



■ **Figure 2.5** Wireframe of mobile configuration screen

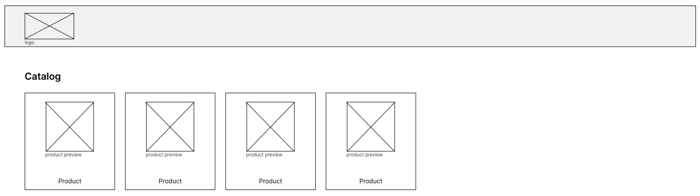


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque quis odio leo. Sed semper, ipsum sed bibendum gravida, lectus est bibendum risus, vitae consequat nisl purus vel elit. Praesent varius, orci nec lacinia dignissim, dui massa ornare massa, in ultrices velit nibh sed dui. Mauris nec dui non dolor auctor consectetur nec eget enim. Nunc at imperdiet nunc. Donec vitae lorem nec augue rutrum eleifend quis eget lorem. Etiam porta convallis libero. Mauris ac elit imperdiet, blandit quam in, venenatis eros.

■ **Figure 2.6** Wireframe of mobile configuration screen with panels

the tool is maintained for all viewport sizes.

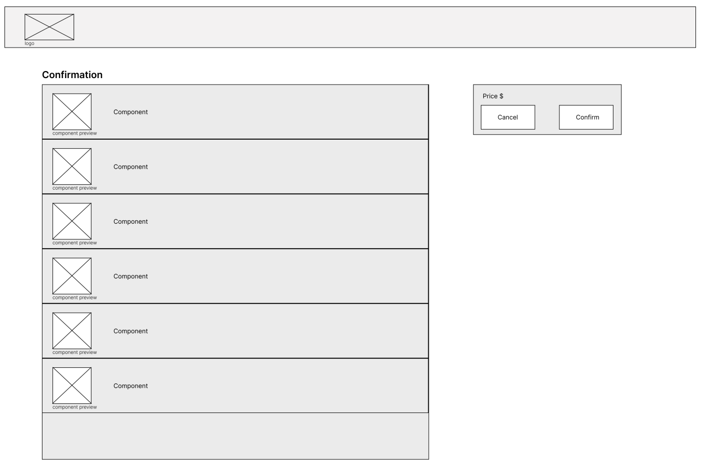
2.3.2 Introduction Screen



■ **Figure 2.7** Wireframe of introduction screen

The introduction screen is the first screen presented to the user when launching the application. It offers a simple way of selecting the configurable product from the catalog, with image and name of the product presented on a large tile. The selection of the product takes the user to the configuration process. Mobile interface of this screen mirrors the large version. The wireframe of the design can be seen in Figure 2.7.

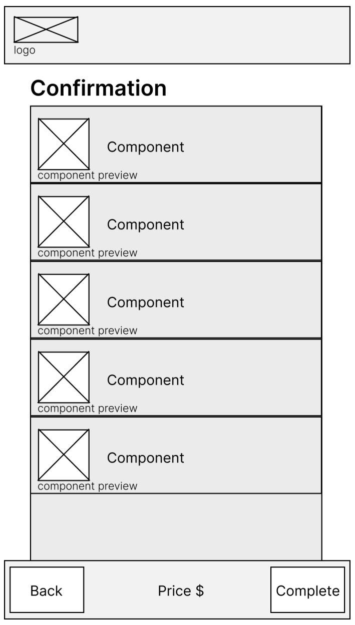
2.3.3 Confirmation Screen



■ **Figure 2.8** Wireframe of confirmation screen

The confirmation screen is presented to the user at the end of the configuration process and satisfies the configuration review requirement (see requirement F9). The wireframe of the configuration screen is shown in Figure 2.8. On the left side, the confirmation screen presents the selected and configured components in a comprehensive list and allows the user to review their choices. The right side of the screen contains two buttons, one to confirm the configuration, which can initiate a confirmation action, and another to return to the configuration process for any adjustments.

The wireframe for the mobile version of the confirmation screen is depicted in Figure 2.9. In this layout, the summary spans the entire width of the screen, with the buttons moved to form a bar at the bottom of the screen.



■ **Figure 2.9** Wireframe of mobile confirmation screen

Chapter 3
Implementation

Architectural structure and data schemas of the application, a comprehensive account of the implementation of core features, preview of the implemented views and evaluation of the requirements.

The following chapter will describe the implementation of the toolkit as envisioned in this thesis, building on the foundation created in the previous analysis and design chapters.

Implementation focuses primarily on the requirements outlined in Subsection 1.2.1, with an emphasis on those of the highest priority. The output of this chapter is the realization of a viable and usable tool.

3.1 Structure of the Toolkit

The application is designed as a front-end only solution. Given it's front-end nature, the configuration's administration can be done through reading from static configuration files rather than relying on data from a backend API. This way, operators of the application can easily adjust the application's behavior and offerings without the need for complex back-end processing, which is not needed in this case as the data are not sensitive or time specific and have a predefined structure.

Consequently, the toolkit is divided into two main applications: the configurator application and the administrator application. The configurator is the user-facing part that businesses deploy or embed on their webservers, offering the ability to configure products based on the supplied configuration files. The administrator application, on the other hand, is used by the toolkit administrators to generate and edit these configuration files.

It should be noted that with this architecture, deploying the administrator application is not a prerequisite for every instance of the configurator; it can be used across various configurators of the same version or even run for the necessary short moment in development mode. This flexibility ensures that businesses can manage and update their configurator setups without the need to deploy an instance of the administrator application.

This architecture not only ensures the toolkit's scalability and ease of use, but also allows to be easily integrated into a wide range of business environments, addressing different customization needs without the complexities and maintenance overheads associated with traditional back-end-dependent applications, avoiding incurring additional costs or technical challenges.

This structure is also implemented in such a way that if the need for a back-end develops at some point, it will be very easy to modify the application for that change.

3.1.1 Project Directory Tree

```
/
├── apps/
│   ├── admin/
│   │   ├── src/
│   │   ├── index.html
│   │   ├── vite.config.ts
│   │   └── package.json
│   └── main/
│       ├── src/
│       ├── index.html
│       ├── vite.config.ts
│       └── package.json
├── packages/
│   └── shared/
│       ├── src/
│       └── package.json
└── package.json
```

■ **Figure 3.1** Directory structure of the project

Given the separation of the toolkit into the configurator and administrator components, the project is split into two subprojects in the `apps` directory. The configurator application is located in the `main` directory, while the `admin` directory contains the source files for the administrator application. Consequently, each of the subprojects has its own Vite.js configuration, among their `index.html` files, `src` directories for code and `package.json` for managing the subproject-specific dependencies.

To minimize code duplication, a shared library is located in the `packages` directory. This shared library contains common elements imported by both applications, such as data schemes, generic React components, custom React hooks, types and interfaces, CSS styles, or other utility functions. Using this shared package in the configurator and administrator applications ensures consistency and reduces redundancy across the toolkit.

Most dependencies are managed centrally in the root `package.json` file, with dependencies specific to the subprojects being handled in their corresponding package files. This approach optimizes dependency management across the entire project.

It is important to note that the presented directory tree focuses on illustrating the separation of the project into subprojects: configurator and administrator applications, and the shared library. It is not exhaustive, as the actual project contains various other config and dot files.

Figure 3.2 presents a brief overview of the `src` directories, which organize the source code into logical sections. The directories have the same layout across all subprojects.

```
src/
├── components/ ........................ UI elements; React components
├── configurations/ .................... configuration of the application
├── hooks/ ........................................ custom React hooks
├── interfaces/ .................................... object interfaces
├── schemas/ ....................................... Zod data schemas
├── stores/ ............................ Valtio data stores and actions
├── styles/ ................................................ CSS styles
├── toasts/ .................................. presets for notifications
└── utilities/ ........................... helper functions and classes
```

■ **Figure 3.2** Structure of `src` directories

3.2 Data Schemas

The data schemas are derived from the domain model discussed in Section 2.2. These schemas define the shape of the objects that are used to transfer and store information within the tool.

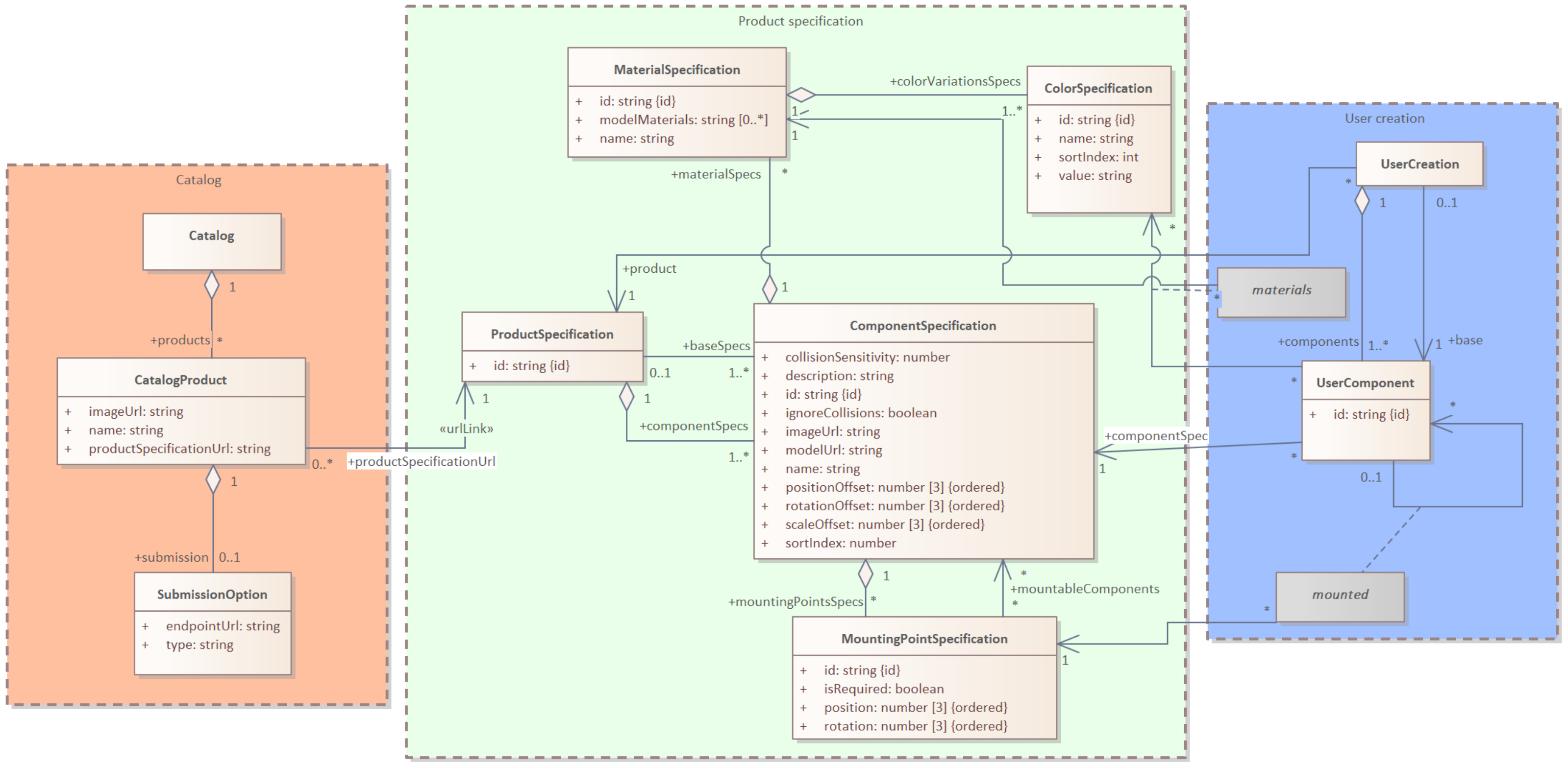
To visualize these schemas, UML diagram is provided in Figure 3.3. The diagram provides a general overview of utilized structures and associations; however, the implementation requires slight adjustments.

Data schemas are stored in the shared library discussed in the previous section. This ensures consistency, as they are used in both the configurator and administrator applications.

The schemas are organized into three separate parts: catalog, product specification, and user creation. Each of these parts corresponds to a separate file in the shared library, in addition to their different roles within the application.

The parts are discussed in the following sections, and the UML diagram in Figure 3.3 provides color coding to differentiate them.

As discussed in Subsubsection 2.1.5.4, the implementation of data schemas utilizes the Zod validation library, which allows straightforward validation of the shape and values of the data during parsing. TypeScript types are then easily created from these Zod schemas, which is illustrated in Code listing B.3 in Appendix B. [56]



■ **Figure 3.3** Data schemas as UML diagram

3.2.1 Catalog

The catalog data schema, visualized with an orange tint in the UML diagram, is essential for the initial user interaction with the application.

```
export const SubmissionOptionSchema = z.object({
  type: SubmissionTypeSchema,
  endpointUrl: z.string().url(),
});

export const CatalogProductSchema = z.object({
  name: z.string().max(100),
  productSpecificationUrl: z.string(),
  imageUrl: z.string(),
  submission: SubmissionOptionSchema.optional(),
});

export const CatalogSchema = z.object({
  products: z.record(CatalogProductSchema),
});
```

■ **Code listing 3.1** Data schema of catalog

Building upon the domain model, the catalog is created by the administrator of the tool and covers all product specifications. However, compared to the domain model, there is an additional level of abstraction in the form of `CatalogProduct`, which contains the most important information about the product specification, such as the name and the preview image. The catalog is separated from the product specifications, which are stored elsewhere. One reason for this is that the catalog will first be fetched when the configuration application is launched, and this information must be presented to the user as soon as possible. The specification of the product, which contains the information needed for the configuration process, can then be fetched from a URL stored inside this object after the user decides which product to configure, which is represented by the stereotype `urlLink` in the UML diagram.

In addition, information about the confirmation action is stored for each product in the form of `SubmissionOption`, which specifies the type of action and the endpoint to which the application potentially sends the confirmed configuration.

The TypeScript implementation can be seen in Code listing 3.1. In the catalog, each product is stored using a record structure, with the ID of the product as a key. The names have been arbitrarily limited to 100 characters to ensure that they can fit within the correct user interface position. URLs, except for the endpoint, are not validated to accommodate local addresses.

3.2.2 Product Specification

The advantage of the structure where the catalog is separate from the specifications, which are fetched only when needed, is that each specification can be sourced from a different location. The specifications can also be updated independently on the rest of the catalog.

The product specification part is visualized by a green tint in the UML diagram.

```
export const ComponentSpecificationSchema = z.object({
  name: z.string().max(100),
  description: z.string(),
  imageUrl: z.string(),
  modelUrl: z.string(),
  materialSpecs: z.record(
    MaterialSpecificationSchema),
  mountingPointsSpecs: z.record(
    MountingPointSpecificationSchema),
  positionOffset: z.tuple(
    [z.number(), z.number(), z.number()]).optional(),
  rotationOffset: z.tuple(
    [z.number(), z.number(), z.number()]).optional(),
  scaleOffset: z.tuple(
    [z.number(), z.number(), z.number()]).optional(),
  ignoreCollisions: z.boolean().optional(),
  collisionSensitivity: z.number().min(50).max(100).optional(),
  sortIndex: z.number().optional(),
});

export const ProductSpecificationSchema = z.object({
  baseSpecs: z.record(z.string()),
  componentSpecs: z.record(ComponentSpecificationSchema),
});
```

■ **Code listing 3.2** Data schema of product and component specifications

This part of the data is created by the tool's administrator and provides information about the available options for user configuration of a chosen product.

The schemas are also derived from the domain model, although with some modifications such as changes in attributes and some entities being merged.

Most of the concepts in this part are identical to the domain model; however, the new concept of `base` component is introduced here. Due to the modularity principle, each component is in some way mounted on another component.

Base components are such components that can initiate the configuration, as they are not mounted to other components. They represent the potentially first component of a configuration. These base components are defined in the product specification.

In the data schema, the model is directly included in the component specification, with the `modelUrl` attribute pointing to a location with the 3D model file. To compensate for this coupling, attributes that modify the position, rotation, and scale of the model are added to the component specification. The component specification, as well as the color specification, include the `sortIndex` attribute, which determines the presentation order to the user, with lower numbers having higher priority. Further attributes are also present, such as description or the option to ignore collisions for this component.

The material specification includes the names of the affected meshes in the attribute `modelMaterials`, and the color specification includes the color value.

In TypeScript, the aggregations depicted in the UML diagram are implemented as records, with the IDs serving as keys and the objects as values. Other associations are implemented using solely IDs. Color values are represented using hex codes. The implementation of the product and component specifications can be viewed in Code listing 3.2. Other entities in this section are implemented similarly and, therefore, do not require detailed code previews within the text of the thesis.

3.2.3 User Creation

```
export const UserComponentSchema = z.object({
  componentSpec: z.string(),
  materials: z.record(z.string()),
  mounted: z.record(z.string()),
});

export const UserCreationSchema = z.object({
  product: z.string(),
  base: z.string(),
  components: z.record(UserComponentSchema),
});
```

■ **Code listing 3.3** Data schema of user creation

`UserCreation` corresponds to the configuration entity of the domain model but has been renamed in the implementation, as “configuration” is an ambiguous term that can mean several things in application development (e.g. configuration of the application itself). This naming clarifies what this data schema represents.

The user creation is utilized by the configurator application for storing chosen options of the user, and is tinted blue in the UML diagram.

The user creation is associated with the product specification and stores its configured components, as well as the selected base component, which is the first component of the configuration. All other components are mounted on the base component or other components.

In contrast to the domain model, the configured component is represented by `UserComponent`. This user component links to a component specification that serves as a template. It also stores information about the mounted components, including the specific mounting points where each component is attached, as well as the selected colors of the materials that have been customized.

In the TypeScript implementation, which can be seen in Code listing 3.3, the user components are stored in a record with their unique ID as the key. The component specification for each user component (as well as the product specification to which the components belong) is linked by its ID. Information about mounted components is stored in records, with the mounting point specification ID as the key to identify the position and the ID of the user component as the value. Similarly, materials are organized with the material specification ID as the key and the ID of the selected color specification for that material as the value.

3.3 Challenges and Solutions

During the implementation phase of the project, various challenges emerged. This section touches on these challenges, detailing both the issues encountered and the innovative solutions used to overcome them. The majority of these challenges were related to the implementation of the defined requirements. Examining these obstacles offers valuable insight into the software engineering involved in the development of product configurators.

3.3.1 Component Visualization

The implementation of component visualization primarily stems from requirement F1, which mandates a 3D visualization of the configured product while employing realistic models. The solution is shaped by the chosen 3D technology (see Subsection 2.1.2), primarily the React Three Fiber library supported by Three.js, and the created data scheme described in the previous section. This also facilitates the open navigation requirement (F5).

A simplified version of the implementation is illustrated in Code listing 3.4. The group feature of the Three.js library is utilized for effective position and rotation composition. The component is primarily made up of this group, which integrates the model and the mounting points of the component.

Inside `ComponentModel`, the 3D model is loaded from a gLTF file, a format which can fully represent 3D objects widely used in web applications and greatly

```
export const Component = ({ componentId }) => {
  const {
    component, componentSpec
  } = useComponent(componentId);

  return (
    <group name={componentId}>
      <ComponentModel componentId={componentId} />

      {Object.entries(componentSpec.mountingPointsSpecs).map(
        ([mountingPointSpecId, mountingPoint]) => {
          const mountedComponentId = component
            .mounted[mountingPointSpecId];

          return (
            <group
              key={mountingPointSpecId}
              position={mountingPoint.position}
              rotation={mountingPoint.rotation}
            >
              {mountedComponentId ? (
                <Component componentId={mountedComponentId} />
              ) : (
                <MountingPointButton
                  componentId={componentId}
                  mountingPointSpecId={mountingPointSpecId}
                />
              )}
            </group>
          );
        }
      )}
    </group>
  );
};
```

■ **Code listing 3.4** Preview of component visualization implementation

supported in the Three.js library. The position, rotation and scale of the model is adjusted by the values stored in the component specification, using a nested R3F group, which can be previewed in Code listing 3.5.

Considering that configured components (other than the base component) are mounted on top of each other, a recursive approach is employed to manage this hierarchical structure and mapping between the data schema and the visualization. Therefore, for each mounting point, a nested group is utilized to adjust the position and rotation according to the mounting point specification. If a component is attached to this mounting point, it is recursively inserted into the group; otherwise, a spatial button is displayed at this position to facilitate the addition of components.

This recursive nesting ensures that each mounted component's position and rotation are automatically adjusted through the group composition, eliminating the need for direct calculation of these values. The simplification this offers significantly outweighs the minor performance impact introduced by recursion.

Properties of components are managed through Valtio proxies using a custom React hook `useComponent`, therefore, whenever the underlying state changes, the interface, including the 3D elements, immediately redraws. All 3D elements are placed within a R3F `canvas` element. The orbital camera controls needed by the requirement F2 are accomplished using the `OrbitControls` Three.js element, integrated using the Drei library, which is placed within the canvas and offers standard orbital camera controls.

3.3.1.1 Component Interactivity

To meet the requirement F4, which enables users to select components directly within the visualization, an `onClick` event is utilized on the group containing the model. Upon clicking on the model, a side panel is presented that displays detailed information about the selected component. To visually highlight the selected component in the visualization, an outline of the model is created using the `Outlines` element from the Drei library, which is included within the same group as the model. The color of the outline can be configured by the application administrator. Additionally, an `onPointerMissed` event employed on the canvas allows to deselect the component by clicking on the background.

3.3.2 Model Material Change

To fulfill the requirement F8, which allows users to modify the appearance of materials of the component, there must be a mechanism that allows to change the colors of parts of the model within the preview. Each gLTF file with model contains meshes and default materials. When the user changes the color of a material, the model needs to be decomposed into these atomic parts. This allows for the material of each part to be found and adjusted.

In the implementation, which is in a simplified form presented in Code listing 3.5, it is iterated in a recursive manner through all meshes of the model, replacing the default materials for the customized ones whenever needed.

As a result of this approach, the model is not inserted into the scene at once, but instead every mesh is included separately along with the corresponding material. This is achieved also using the group where all the meshes of the model reside, which, as mentioned in the previous section, changes the position, rotation, and scale of the model.

Changing the colors of the materials is facilitated through the side panel. The new color value is stored in a Valtio proxy, from which it is read when composing the 3D model, ensuring dynamic updates of the visual preview whenever the user makes changes.

```
export const Render = ({
  object,
  materialOverride
}: RenderProps) => {
  if (object.type === "Group" || object.type === "Object3D") {
    return (
      <group data={object.data}>
        {object.children.map((child) => (
          <Render
            object={child}
            materialOverride={materialOverrides}
          />
        ))}
      </group>
    );
  } else if (object.type === "Mesh") {
    return (
      <mesh
        material={getMaterial(mesh.material, materialOverride)}
        data={object.data}}
      />
    );
  }
};
```

■ **Code listing 3.5** Preview of model composition implementation

3.3.3 Undo and Redo Actions

Undo and redo actions enable users to revert the changes made during the configuration process, fulfilling the requirement F13. Buttons for these actions are intuitively designed as forward and backward arrows and are placed around the preview view.

Since the user's creation is stored in a Valtio proxy during the configuration process, the implementation of this feature is straightforward. Valtio provides an enhanced version of the proxy that maintains a history in the form of snapshots and includes callable undo and redo functions. The management of snapshots is smart, as multiple actions performed immediately after each other are considered to be aggregated within one snapshot. This is particularly useful for this application, since, for example, the addition of a new component consists of two operations: the creation of a new component in the store and then mounting it on a point on another component, which is correctly considered as only one snapshot. Therefore, except for the setup of the initial snapshot, this implementation approach did not need any additional adjustments.

3.3.4 Collision Detection

Collision detection is necessary for preventing impossible configurations by disallowing physically overlaping components. This feature directly addresses the requirement F7. As the shapes of the components can be highly complicated, the method for detecting collisions needs to be based on the 3D preview working directly with the component models.

Three.js provides a bounding-box method (where each model is considered to be cube shaped, and overlaps are determined based on these primitive shapes), which is far too simple for complex shapes and large amounts of models; therefore, a more sophisticated approach was needed.

The three-mesh-bvh[1] package extends the Three.js library by providing an implementation of a Bounding Volume Hierarchy (BVH) tree, which enables efficient shape intersection queries for high-definition meshes. [57]

To compute the bounds tree, the entire scene is wrapped in a `bvh` element from the Drei library, which performs this computation on each mesh. Collision detection is performed for all mountable components at unoccupied mounting point whenever the user attempts an addition of a component or when attempting to swap the configured component. To do so, the model of the potential component is loaded, adjusted by the values in the component specification, and its position is set to the potential mounting location in the scene. This is not visible in the preview to the user and is performed using standard Three.js code.

Then, each mesh of the model is queried on geometry collision against the meshes of models of existing components in the preview, excluding the component containing the mounting point. This exclusion happens because, when a mounting point exists, even if there is a collision with the owning component, the defined mountable component should be allowed to be mounted. In addition, the administrator of the tool can exclude certain components from all collision calculations by specifying this in the component specification. The collision sensitivity can also be adjusted for each component specification. Setting this option to low sensitivity (smaller number) causes the model that is checked for

[1]`https://github.com/gkjohnson/three-mesh-bvh`

collisions with the scene to be slightly reduced in size, meaning that close fits are more likely to pass.

In case the model collides with models of existing components in the scene, the option to add the component to the configuration is not presented to the user.

A highly simplified implementation that illustrates this principle can be seen in Code listing 3.6. Although collision detection is highly computationally intensive, it is performed only after specific user actions, which are not supposed to be very frequent; therefore, the possible performance hit is deemed acceptable given its benefits.

```
export function collisionDetection(
  model: THREE.Group,
  scene: THREE.Scene
): boolean {
  const collisionDetected = false;

  traverseMeshes(model, (modelMesh) => {
    traverseMeshes(scene, (sceneMesh) => {
      const transformMatrix = new THREE.Matrix4()
        .copy(sceneMesh.matrixWorld)
        .invert()
        .multiply(modelMesh.matrixWorld);

      if (
        (sceneMesh.geometry as BVHBufferGeometry)
          .boundsTree.intersectsGeometry(
            modelMesh.geometry,
            transformMatrix
          )
      ) {
        collisionDetected = true;
      }
    });
  });

  return collisionDetected;
}
```

■ **Code listing 3.6** Preview of collision detection implementation

3.3.5 Configuration Processing

For the implemented tool to be useful, there needs to be a mechanism that will facilitate performing further actions with the user's creation when the configuration process finishes. This is in accordance with the requirement F10.

As needed by requirement F9 and discussed previously, after completion of the product configuration, users are presented with a summary screen that details the selected options. To save the configuration for future reference in a readable way, a button has been introduced that enables saving this summary to PDF. Implementation of this button action needed to be done on the front-end and utilizes the HTML print API.

```
export const ContactInfoSchema = z.object({
  name: z.string(),
  email: z.string().email(),
  phone: z.string().max(26).optional(),
  note: z.string().max(10000).optional(),
});
```

■ **Code listing 3.7** Data schema of contact information

The administrator of the tool can define the confirmation action in the catalog. If such an action is defined, it may include an endpoint URL to ensure that it can be executed. The supported confirmation actions are as follows:

1. **Preview**: This is the default behavior if no confirmation action is defined. In this case, the confirmation button is not presented to the user at all, and the configurator serves only as a preview tool. The URL of the endpoint is also not defined.

2. **Redirect**: This action allows for a simple flow where clicking the confirmation button redirects the user to the chosen URL.

3. **Webhook**: Upon clicking the confirmation button, the user configuration, in the form of the `UserCreation` data schema presented in the previous sections, is sent using a POST request to the defined endpoint URL. The user creation is sent as a JSON in the body of the request. In order to present the result of this action as a success to the user, the response must return OK. Optionally, the response can contain a JSON body with valid URL in the `redirectUrl` field. If this is the case, and the request has been successful, users will be redirected to this returned URL, otherwise, the tool returns to the initial view. This mechanic facilitates possible integration with other systems or serverless functions for further processing.

4. **Inquiry form**: This action presents the user with an inquiry form asking for their contact information, such as name, email, phone number, and

additional notes. This fulfills the requirement F11. When the form is sent, the process is identical to the webhook case, with the difference that the request contains the user contact details in the JSON payload under the field `contact`. This allows for direct creation of inquiries without the need for users to navigate away from the application. The structure of the contact data schema is detailed in Code listing 3.7.

3.3.6 Application Configuration

Application configuration is central for setting the behavior and appearance of the tool so that the individual preferences of the operator are met. Given that the solution is designed for adaptability, it is essential to provide extensive setup options. These settings are managed by the administrator of the application, allowing them to tailor the application without the need to change the code directly. This also helps to fulfill the interface customization requirement (F14).

```
fetch(globalConfigUrl)
  .then((response) => {
    return response.json();
  })
  .then((data) => {
    globalConfig.config = AppConfigSchema.parse(data);
  })
  .then(() => {
    const i18n = configureI18n();
    const router = createBrowserRouter(routes);
    root.render(
      <React.StrictMode>
        <App i18n={i18n} router={router} />
      </React.StrictMode>
    );
  })
  .catch((error) => {
    console.error(error);
    root.render(
      <p className="text-red-600">Fatal error</p>
    );
  });
```

■ **Code listing 3.8** Preview of application initialization and config loading implementation

As the tool is front-end-based, its configuration is handled using a JSON file. The file is fetched at the start of the application, before React is even initialized, from location `/appconfig.json`. This ensures that the configuration is not baked in the application, rather it is refreshed each time the application is accessed, meaning that the administrator can update the settings without the need to redeploy the entire application, which supports the maintainability requirement (NF5). The implementation of the application initialization with the fetch of the config can be seen in Code listing 3.8.

This may slightly increase the First Contentful Paint (FCP) metric, which measures the time it takes to display anything on screen during the loading of the application. [58]

However, in this case, the impact is minimal, as the resource is relatively small, and located in the same location as the rest of the application.

Right after the config file is fetched, libraries for routing and internationalization are prepared. Then, React and the rest of the app is initialized.

The config file includes the URL with the location of the catalog (which is therefore also fetched at runtime, allowing for live updates), toggles for features, settings of the 3D preview, and color settings for the tint of the interface, meaning unless larger customization is required, there is no need for CSS modifications. It also includes locations of images such as logos or favicons, as well as the title of the website.

3.3.6.1 Interface Texts

As described in Subsubsection 2.1.5.2, the i18next library was chosen for localization. This addresses the multilingual support requirement (NF8), as well as simplifies the interface text customization requirement (F15).

The library is setup in such a way that it expects for all localization files to be located in `/locales/{language}/translation.json`, where `{language}` is a two-letter language code. This structure allows simple addition of further languages, as the application performs lookup for these translation files. Then the general config file needs to be updated with the list of available languages. Additionally, one language must be designated as the default localization in the general config file, which will serve as a fallback.

Like the general config file, the translation files are loaded at runtime, therefore changes to the interface texts or additions of new localizations can be performed without redeploying of the application.

The application attempts to automatically choose the correct language based on the detected browser settings, but also allows users to override this selection via a two-letter parameter in the `?lng=` query of any URL in the application. The language is also changeable using a menu.

As part of the implementation, a default template localization in English is provided, establishing a baseline.

3.3.7 Routing

Routing within the application is performed using the React-Router library, based on a choice discussed in Subsubsection 2.1.5.1.

Routes are split based on the three main screens designed using wireframes in Section 2.3. The routing pattern is implemented as follows:

- Route `/` navigates to the introduction screen
- Route `/editor/:productId` navigates to the screen with the configuration process itself. The parameter of product ID chooses the product from the catalog.
- Route `/confirm/:productId` navigates to the confirmation screen using the same product ID as in the configuration process.

Routing helps the user navigate the application better, as forward and backward browser actions are possible, meaning that users can return from the confirmation screen to the configuration screen by clicking back; this means that the browsing history is preserved. Navigation between screens occurs without page refreshes, maintaining a seamless user experience.

From an implementation standpoint, the utilization of different screens is also simplified, as they are managed centrally by the router, and are defined declaratively.

Errors are handled by the router as well; therefore, if there is any issue that throws an unrecoverable error during the runtime of the application, it is intercepted by the router and an error screen is displayed.

3.3.7.1 Data Flow

As the application follows a specific flow between the three routes during usage, some data manipulation is closely tied to this process. The data flow is managed by the loader functions that are executed when accessing each route. This is necessary to ensure that the application handles transitions smoothly and maintains data integrity throughout the user's session.

In the introduction screen route, the catalog is fetched into the memory to allow users to choose which product to configure.

The configuration screen route fetches the product specification based on the ID specified within the route and prepares the user creation. In case the introduction screen is not opened and users navigate directly to this route, the catalog is fetched beforehand as well, to ensure that all necessary data are available.

Furthermore, in the confirmation screen route, it is verified that some user creation with this product ID was created. If not, the user is redirected to the configuration screen.

The error screen is displayed if there are any discrepancies during the routing process, such as if the route requests a product that does not exist in the catalog.

3.3.8 Catalog Management

As outlined in the previous sections, the toolkit is structured into two separate applications: the user-facing configurator and the administrator application. This approach aligns with requirement F16, which demands that catalog management should be visual rather than relying on direct file editing, and should mirror the preview visible during the configuration process. Furthemore, this also addresses the requirement F17, so that the properties of the products within the catalog are easily manageable.

The administrator application is organized into two screens, each tailored for specific management tasks, which is reflected in its routing patterns, which are as follows:

- Route `/catalogcomposer` and `/` navigates to a catalog composer screen, which provides an interface for editing the catalog data schema. In essence, it serves to modify the JSON file that contains the catalog using a visually appealing and user-friendly approach. The interface is based on structured forms that, compared to composing the file manually, enhance the efficiency and prevent errors.

- Route `/productcomposer` navigates to a product composer screen that serves to set up component specifications and their properties. The interface of product composer aims to mirror the interface of the configurator application during the configuration process.

For visual editing of positioning within the product composer's 3D preview, Drei's `PivotControl` is used, which displays interactive three-axis arrows for adjusting position, rotation, and scale of the component specifications. Interacting with this control mechanism immediately stores the new values in the underlying Valtio proxy, meaning the changes are immediately reflected in both the 3D preview and the side panel, where the corresponding values are displayed in input fields.

Given that no back-end has been implemented, the administrator application does not feature a save button. Instead, the capability to export the relevant JSON files containing the specified data schemas is provided. In addition, import of these already existing files is also possible, which provides flexibility and eases modifications.

The administrator application is used when administrators need to update product specifications or manage catalog settings. The implementation is done in such a way that future back-end integration tying the configurator and administrator applications should be possible if the need and opportunity arise.

3.4 Views

This section presents the visual aspect of the implemented views and screens, which are based on the wireframes described in Section 2.3. The implemented interfaces are shown using screenshots, providing a visual confirmation of the design's adherence to the designed wireframes and demonstrating the application of the design concepts discussed earlier.

The choice of technologies (such as TailwindCSS) was paramount to this task, as it greatly facilitated the straightforward implementation of responsive design, making sure that the application performs well across a variety of device sizes and resolutions, and supports accessibility features such as automatic support of dark mode.

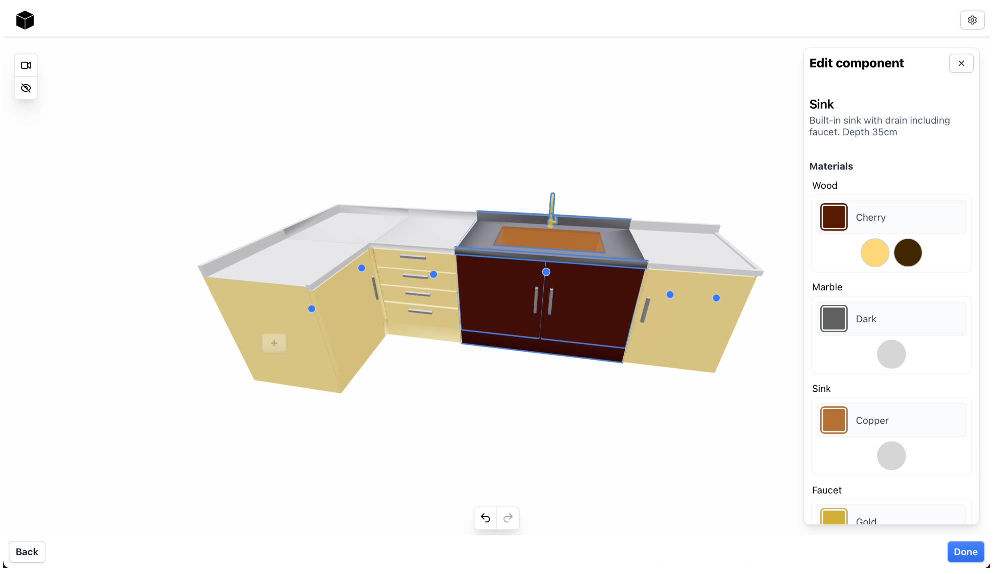
3.4.1 Configurator Application

In the following section, screenshots of the three screens that were described using the wireframes are displayed. The interface is shown only in its light-mode version, given that the dark mode is identical, only with an inverted color scheme.

For demonstration purposes, the screenshots feature a sample product loaded from the catalog. The top bar displays a fictional logo of a business, which is clickable and leads to a homepage of the business, adjustable in the application's settings.

3.4.1.1 Configuration Screen

Screenshot of the implemented configuration screen is presented in Figure 3.4.


Edit component

Sink

Built-in sink with drain including faucet. Depth 35cm

Materials

Wood: Cherry

Marble: Dark

Sink: Copper

Faucet: Gold

Figure 3.4 Screenshot of configuration screen
Source: 3D assets used within the interface are adapted from [59]

Compared to the wireframes, the top bar also includes a settings button on the right, allowing users to choose theme or language.

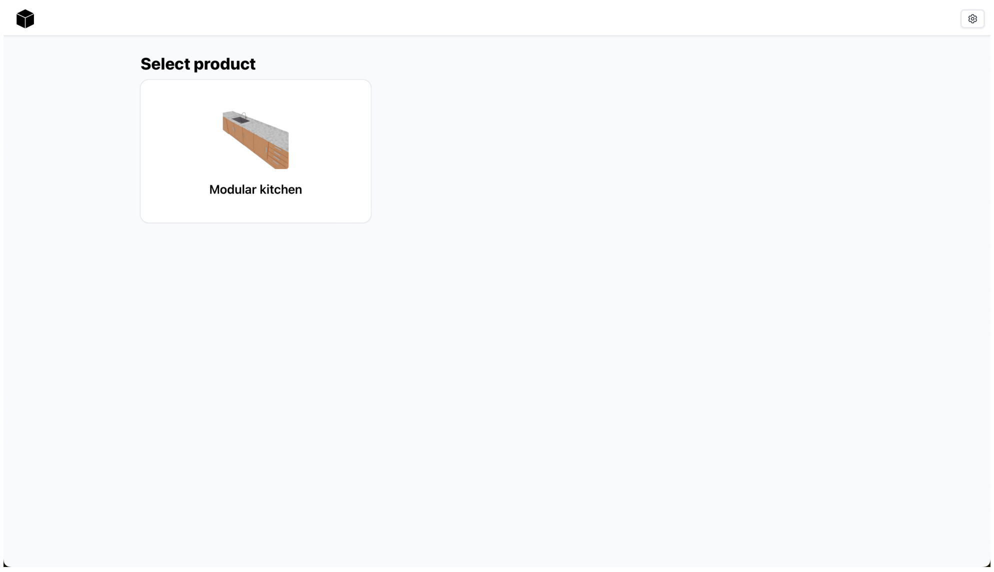
The screenshot captures the interface during the ongoing configuration process with sample products, illustrating the layout that includes an expanded side panel. The 3D preview displays the components that have been configured by the user, along with spatial buttons for adding additional components. The selected component is highlighted using blue edges.

The side panel shows more details about the selected component, along with the options to change the color of its materials, remove it, or change it.

Buttons located in the upper left corner of the preview allow for resetting the camera angle or hiding spatial buttons, while the buttons at the bottom center are dedicated to undo and redo actions. This is a slight deviation from the wireframe, as placing the important action buttons here seems more convenient than in the upper left.

3.4.1.2 Introduction Screen

The implemented introduction screen is presented in Figure 3.5. The implementation adheres closely to the designed wireframe. The background color of the main section differentiates this part from the top bar and enhances the visibility of the tiles.



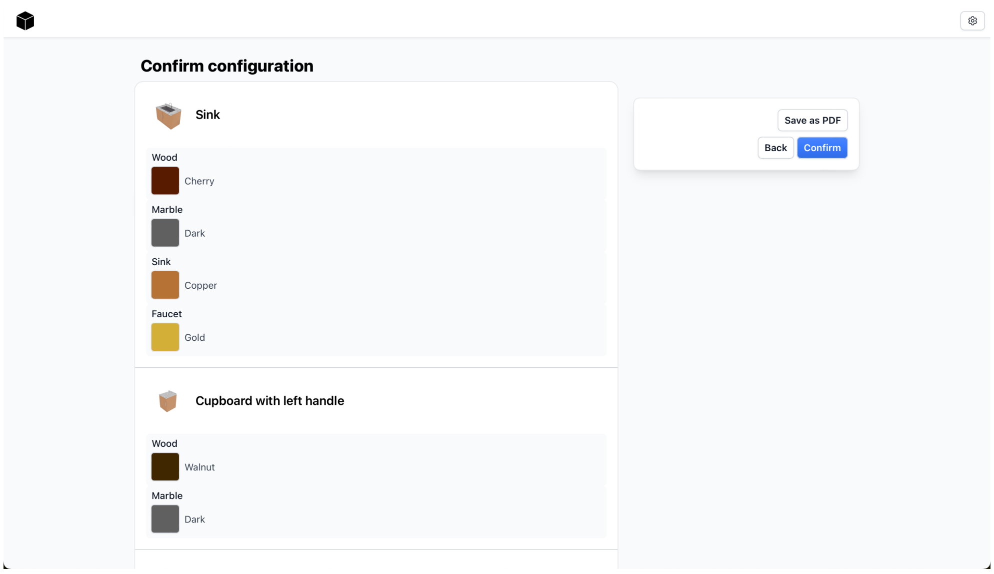
■ **Figure 3.5** Screenshot of introduction screen
Source: 3D assets used within the interface are adapted from [59]

3.4.1.3 Confirmation Screen

The implementation of the confirmation screen again closely follows the designed wireframe, as shown in Figure 3.6.

The screenshot contains a list of configured components for a sample product, optionally displaying the details in the form of selected colors of their materials.

The panel on the right side, which contains the different buttons, is floating; therefore, it stays fixed in the same position even if the page is scrolled. The panel contains back and confirm buttons, as well as a button for printing the component list visible on the left.



Confirm configuration

Sink

Wood
Cherry
Marble
Dark
Sink
Copper
Faucet
Gold

Cupboard with left handle

Wood
Walnut
Marble
Dark

■ **Figure 3.6** Screenshot of confirmation screen
Source: 3D assets used within the interface are adapted from [59]

3.4.2 Administrator Application

This section presents the interfaces of the administrator application, the functionality of which is described in more detail in previous sections (see Subsection 3.3.8).

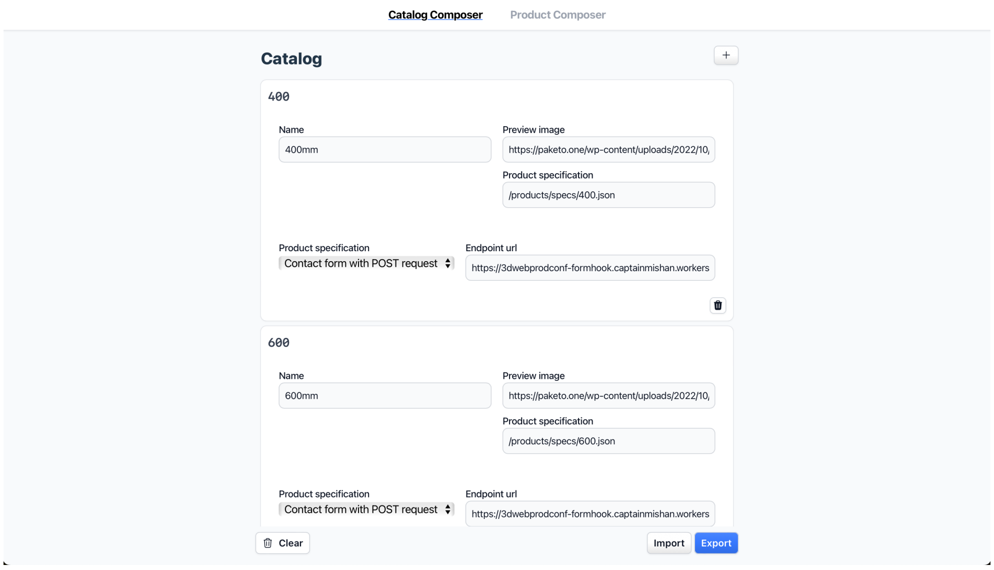
For demonstration purposes, the screenshots included here feature sample products, which are being prepared for use in the configurator application.

The top bar in the administrator application includes a toggle switch between the two screens: catalog composer and product composer.

3.4.2.1 Catalog Composer

The screen presents catalog entries as tiles, each containing a form with editable fields of the catalog product. These entries can be easily added using a button on the top right, and removed using a button within the tile or a button that removes all catalog products located at the bottom.

The catalog can be downloaded as a JSON file using an export button located at the bottom of the screen. Furthermore, the catalog can be imported



Catalog Composer Product Composer

Catalog +

400

Name: 400mm

Preview image: https://paketo.one/wp-content/uploads/2022/10,

Product specification: /products/specs/400.json

Product specification: Contact form with POST request

Endpoint url: https://3dwebprodconf-formhook.captainmishan.workers

600

Name: 600mm

Preview image: https://paketo.one/wp-content/uploads/2022/10,

Product specification: /products/specs/600.json

Product specification: Contact form with POST request

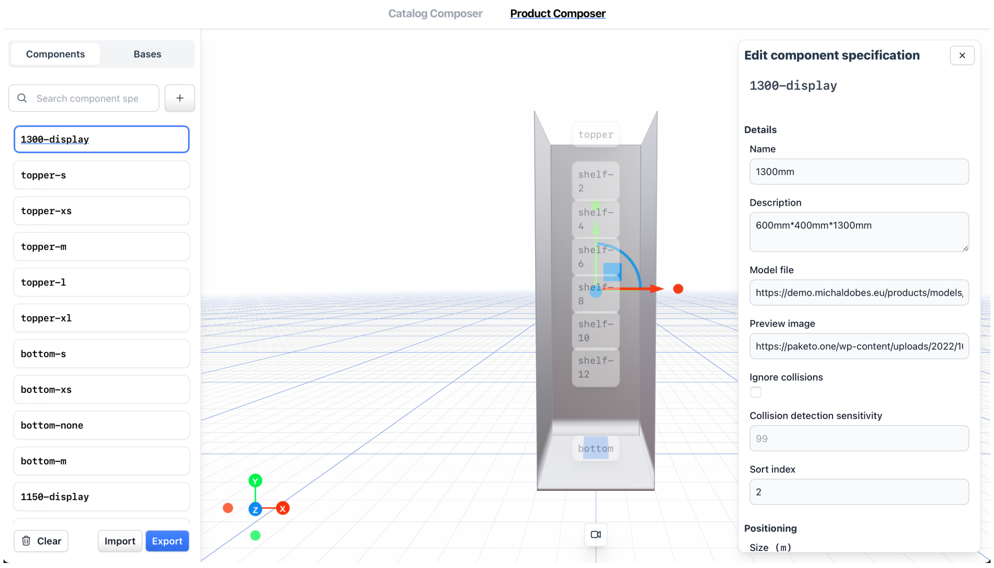
Endpoint url: https://3dwebprodconf-formhook.captainmishan.workers

Clear Import Export

Figure 3.7 Screenshot of catalog composer screen

from JSON using an import button. The exported file can then be utilized in the user-facing configurator application.

3.4.2.2 Product Composer



Catalog Composer Product Composer

Components Bases

Search component spe +

```
1300-display
topper-s
topper-xs
topper-m
topper-l
topper-xl
bottom-s
bottom-xs
bottom-none
bottom-m
1150-display
```

Clear Import Export

Edit component specification ×

1300-display

Details

Name: 1300mm

Description: 600mm*400mm*1300mm

Model file: https://demo.michaldobes.eu/products/models,

Preview image: https://paketo.one/wp-content/uploads/2022/1(

Ignore collisions: ☐

Collision detection sensitivity: 99

Sort index: 2

Positioning

Size (m)

Figure 3.8 Screenshot of product composer screen

The product composer screen, visible in Figure 3.8, copies the layout of the configuration screen of the user-facing application, but has an additional side panel on the left. This panel is used to add and remove component specifications or to select the potential base component specifications of the product specification. Bottom of this panel also contains the clear, import and export buttons, which serve to download or upload product specifications in a JSON file. Selecting a component specification within this panel displays its model in the 3D preview area, while the right side panel, used during the configuration process to present detailed information, allows for the adjustment of all component specification attributes, such as model file, name, position, rotation, scale, mounting points, and materials.

The 3D preview also includes helpful features, such as a navigation gizmo indicating the current orientation of the camera, or grid that serves as a reference plane under the model.

3.5 Summary of Implementation

This chapter outlined the successful implementation of a viable toolkit for the online configuration of 3D modular products. The implementation fulfills most of the requirements set out in Subsection 1.2.1, including all with the highest priority.

Features	**Implemented solution**
Platform	Web
Navigation	Open
Visualization	Realistic
Placement options	Fixed points
Camera movement	Orbital
Impossible configurations	No[2]
Responsiveness	Yes
Price calculation	No
Purchase option	Webhook; Inquiry form; Redirection
Save option	PDF only
Version history	Undo & redo
VR or AR	No
Real dimensions	No

Table 3.1 Summary of features implemented in the configurator application

The implemented versatile toolkit provides robust solutions for both user and administrative needs. Users can flexibly configure modular products through an intuitive interface, while administrators are provided with the necessary tools to efficiently handle product specifications.

[2]Depends on the setup of the product specification

	Requirement	Fulfilled
F1	3D product visualization	Yes
F2	Dynamic orbital camera controls	Yes
F3	Modularity configuration mechanism	Yes
F4	Component interactivity	Yes
F5	Open navigation	Yes
F6	Fixed point component placement	Yes
F7	Component collision detection	Yes
F8	Material color configuration	Yes
F9	Configuration review	Yes
F10	Configuration processing	Yes
F11	Inquiry form	Yes
F12	Configuration saving and retrieval	No
F13	Undo and redo actions	Yes
F14	Interface appearance customization	Partially
F15	Interface texts customization	Yes
F16	Visual catalog management	Yes
F17	Product properties and attributes management	Yes
F18	Real-time price calculation	No
F19	AR viewing capabilities	No
F20	Parametric configuration mechanism	No
F21	Real dimensions visualization	No

Table 3.2 Summary of fulfilled functional requirements

The implemented functional requirements are summarized in Table 3.2, providing a clear overview of the project's achievements and the functionality of the toolkit. As shown, the majority of the requirements have been implemented.

Notably, the requirements concerning the configuration saving and retrieval (F12), real-time price calculation (F18), AR viewing capabilities (F19), parametric configuration mechanism (F20) and real dimensions visualization (F21) have not been implemented. These areas represent potential opportunities for further development. The omission of these functionalities was due to prioritization based on the scope of the project and the availability of resources. Current technological constraints also played a role in the lack of implementation of the AR viewing capabilities.

The requirement for interface appearance customization has (F14) has been partially satisfied, as the application allows to change the color scheme by changing the config file, but more complex elements, such as button style are customizable only by changing directly the CSS code.

Complementing the functional requirements summary is a comprehensive overview of the set of features implemented in the solution, presented in Table 3.1. The summary follows the same structure as used in the analysis of existing product configurators, which can be seen in Table 1.1. However, compared to the existing configurators, it is important to keep in mind that the tool developed here is different in two ways: it is meant to be adaptable and product-agnostic, therefore it comprises of an admin application that allows different types of products to be set up. In addition, it focuses on cost-effectiveness and simplicity. Despite this, the tool still provides features comparable to those of existing solutions deployed by companies.

3.5.1 Possible Future Improvements

There are various possible enhancement this solution could incorporate to enhance the user experience and extend its capabilities.

- **Implementation of missing functionalities**: Some of the non-priorized requirements, such as local configuration saving and retrieval, real-time price calculation, and augmented reality viewing, were not implemented due to resource constraints and technological limitations. Future work could focus on integrating these features.
- **Back-end integration:** While the toolkit currently operates primarily on the client side, developing a corresponding back-end could increase the possibilities of complex features such as better data management, user account management and better integrations with other services.
- **Full rules evaluation:** Adding a rule evaluation engine would increase the amount of products that this toolkit is suitable for, as the handling of sophisticated product configurations may involve multiple dependencies and conditions, which this feature could ensure.
- **Integrations**: Integrating the application to existing e-commerce platforms, such as Shopify, WooCommerce, or Shoptet, could significantly expand the toolkit's usability. This integration would allow data synchronization between the configurator and the e-commerce platforms, enabling automated updates to product listings or a direct checkout process.
- **UI Themes:** The user interface currently has customizable color scheme; however, larger customizations need to be made by editing the CSS code. In the future, there could be more predefined themes of the interface, which the operator of the toolkit could choose from.
- **User interaction improvements**: Additional interactivity features could be implemented, such as drag-and-drop addition of components to the configuration, or providing detailed information panels or tooltips for components.

The enhancements described in this section outline possible future development directions of this toolkit, aimed at increasing functionality, improving user experience, and ensuring scalability.

Chapter 4
Deployment

Example deployment, configuration and customization of the application. Examination of associated costs and the impact on business processes.

In the following chapter, the attention is shifted from the development process to the deployment of the created solution. In this chapter, the focus is two-fold: firstly, the deployment of the solution is described in practical steps from the toolkit's administrator perspective, then business implications such as cost and process optimization are examined.

The setup of the solution is described from the point of view of a business wanting to utilize the tool. For illustration purposes, an example company is used in this chapter. The chosen company manufactures modular point-of-sale cardboard displays. To offer their product, they maintain a basic online presence using a simple WordPress website with information about the company, a showcase of their products, and an inquiry form which allows them to be contacted by their customer. In this scenario, the company aims to integrate the 3D configurator into its existing website to offer an interactive, user-friendly service that allows customers to customize and visualize options for their own configurations of cardboard displays.

By the end of this chapter, a comprehensive understanding of the deployment process of the toolkit from both a technical and business perspective will be provided.

4.1 Application Setup and Configuration

This section describes the necessary steps needed to utilize the implemented solution, from the perspective of a person designated by the business to operate the toolkit.

4.1.1 Building and Launching the Application

If the code of the application has been modified in any way, the installation of Node.js and npm is a prerequisite for building the application. The configurator application can then be built by executing `npm install` followed by `npm run build --workspace main`, which will generate the files in `apps/main/dist/` directory. If no modifications have been made, pre-built files may be utilized directly.

In the example case, the company already operates a website; therefore, the newly implemented configurator tool will be deployed to the appropriate subdomain of the website (e.g. `configurator.example.com`). This approach requires proper configuration of DNS settings for the new subdomain and appropriate adjustments to the web server, but otherwise utilizes the already existing web hosting infrastructure. If this were not the case, it would be necessary to facilitate web hosting through an HTTP server.

To deploy the application, the built files need to be copied to the web server, ensuring that they are statically served. The `index.html` file loads and initializes that application on access.

Due to the implementation of client-side routing, the web server needs to be adjusted to correctly redirect requests. A web server by default interprets request as a query for files located on the URL address, and since the routes defined in the application do not correspond to the actual files, the "not found" responses would be returned. Therefore, the server needs to be configured in such a way, that requests for nonexistent files are redirected to the `index.html` file, where they will be managed internally by React-Router. The company in the example case uses Apache HTTP Server, therefore its configuration `.httaccess` file, which sets the rules such that requests for nonexistent files, directories, or symlinks are redirected to `index.html` can be previewed in Code listing 4.1. A similar setup can be achieved in comparable fashion with NGINX or other web servers.

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l

RewriteRule ^ index.html [L]
```

■ **Code listing 4.1** Configuration of Apache HTTP Server for client-side routing

With this set, the application should be functional and available, albeit yet without the content of configurable products.

4.1.2 Customizing the Application

The following section describes the adaptation of the deployed application to correspond to the identity and meet the needs of the business. This is done by the administrator through modifications of the `appconfig.json` file.

4.1.2.1 Localizations

To adjust the interface texts and provide different localization options, translation files should be created in the `locales/` directory. This functionality is described in more detail in Subsubsection 3.3.6.1. These translations can be based on the English translation file included with the built application. Once the necessary language files are set up (note that supporting English is optional), they should be listed by their two-letter codes in the `appconfig.json` file under the `ui.languages.all` key. Additionally, a default language must be specified in `ui.languages.default`.

4.1.2.2 Appearance

Following, the appearance of the interface can be tailored. Color scheme is customizable by updating the hex color codes in `ui.colors` and for the outlines within the 3D preview in `spatialUi.selectionColors`. The logo displayed on the top left, as well as the favicon can be customized by setting the paths to the image files under `images` key. The web page to which the displayed logo links to can be configured using the `sources.homepageUrl` key. In addition, the title of the web page can be adjusted using the `title` key. The appearance customizations provide options for both dark and light modes of the interface.

Moreover, the camera style and floor shadow in the 3D visualization can be adjusted, and the option whether to display the button to save the completed configuration in PDF to users is provided.

4.1.2.3 Data Source

Finally, the source from which the product catalog is fetched must be defined in `sources.catalogUrl`. Any accessible location which provides the correct data scheme can be provided here. The creation of this catalog file is described in the following sections.

4.1.2.4 Example Case

In the example case discussed in this chapter, the interface was customized according to the company's brand guidelines. The company logo, set to appear in the top left corner, links back to the company's original website. The catalog source was defined as a static file located on the web server at `products/catalog.json`.

4.1.3 Catalog Creation and Content Management

The following section describes the process of managing configurable products for the deployed application.

The first step needed to create the catalog of products is the preparation of 3D models that will be used in the configurator. The supported format is gLTF and its binary form GLB, to which other 3D formats can be converted. These files can be generated out of Computer-Aided Design (CAD) files used by the company to manufacture the products, or modeled manually. The images of the components that help the models with the representation should also be prepared and kept somewhere accessible.

It is necessary to store the files in an accessible location so that they can be fetched from their paths by the configurator application. It should be kept in mind that the size of the file and the complexity of the models correspond to the performance of the configurator application; therefore, it is beneficial to have simplified models with compressed textures to achieve the greatest possible performance. For easier component specification creation, the center of the model should also align with the center of the 3D scene in the file.

To create the configurable products, the administrator application needs to be launched. With the same prerequisites and dependencies as the main application, the admin tool can be built using the following command: `npm run build --workspace admin`

The resulting files will be created in `apps/admin/dist`. These built files can be again uploaded to a web server (perhaps within the company intranet) or be served locally for the duration needed to create the specifications by executing the command: `npm run preview --workspace admin`

Within the administrator application's product composer screen, product specification can be now created, by creating component specifications along with their materials, mounting points and specified base components. It should be kept in mind that the 3D models of the components (and images) in the administrator application are loaded from the same paths as they will be in the configurator application, and it is therefore crucial for these models to be accessible in both applications on the same paths. When product specifications are created, they can be exported to JSON files, and again need to be stored in locations accessible from the deployed configurator application.

The catalog is then created in the catalog composer screen of the administrator application. Each catalog entry must reference the location accessible from the deployed configurator application where the product specification file created in the previous step can be fetched from. After the catalog is complete, it should be exported and stored at the location specified in the global config file, which was discussed in previous section.

For updates of the created product specifications and the catalog, the existing files can be downloaded, imported back into the administrator tool, edited, and then uploaded back to their original locations.

4.1.3.1 Example Case

```
async function handleRequest(request: Request) {
  const headers = new Headers({
    "Access-Control-Allow-Origin": "configurator.example.com",
    "Access-Control-Allow-Methods": "POST, OPTIONS",
    "Access-Control-Allow-Headers": "Content-Type"
  });

  if (request.method === "OPTIONS") {
    return new Response(null,
        { headers, status: 204 });
  }
  if (request.method !== "POST") {
    return new Response("Method Not Allowed",
        { headers, status: 405 });
  }

  const formData = await request.json();
  try {
    const parsedData = RequestSchema.parse(formData);
    await sendEmail(parsedData);
    return new Response(null,
        { headers, status: 200 });
  } catch (error) {
    return new Response("Invalid data",
        { headers, status: 400 });
  }
}

addEventListener("fetch", event => {
  event.respondWith(handleRequest(event.request));
});
```

■ **Code listing 4.2** Implementation of serverless function for forwarding inquiry form data

In the example company followed in this chapter, the 3D models were generated from CAD software used in manufacturing process of the company and stored as static files on the web server. Then, the administrator application was utilized to create the product specifications and the catalog. The created files were also transferred to the web server, from where they will be statically served. Given that the company utilizes an inquiry form on their website, the configurable products followed the same approach and had an inquiry form set as a confirmation action. A serverless function, hosted on Cloudflare Workers[1], was created to process the data. The function accepts POST requests with the contact info and configuration created by the user sent from the inquiry form within the application, and forwards these data to a company email. To enable this, the URL of this serverless function was set as the endpoint of the confirmation action in the catalog composer screen. A simplified illustration of the implementation of the serverless function can be seen in Code listing 4.2.

4.2 Business Aspects

This section explores the business aspects of deploying the configurator tool, examining the cost-effectiveness as well as business process modifications within the company.

4.2.1 Cost-Effectiveness

As this solution aims to cater to small businesses, this project has been conceptualized with cost-effectiveness in mind, with the requirement of lightweight infrastructure needs (NF4) and self-hostability (NF3). These principles have been greatly reflected in the design and implementation.

The primary costs of this solution stem from the setting up of the tool, which includes acquiring 3D models and specifying the configurable content, as well as the customization of the tool and its integration into the infrastructure. These costs can vary greatly between different companies, as some may be disposing of the necessary files, while others will have to create them. The complexity of the offered products that are defined in the tool also plays a role in this step.

From a technical standpoint, the infrastructure costs are low. For companies with an existing web server, no additional infrastructure is required. In addition, the costs of web hosting are minimal, as static websites can often be hosted for free in some capacity on platforms such as Cloudflare Pages [60] or Netlify [61].

Provider	**Free plan limits**	**Cost over limit (per million reqs)**
Cloudflare Workers	100,000 reqs/day	$0.30
AWS Lambda	1,000,000 reqs/month	$0.20
Google Cloud Functions	2,000,000 reqs/month	$0.40

Table 4.1 Overview of the pricing per requests (reqs) for serverless providers
Source: Cloudflare [62], Amazon [63], Google [64]

[1] https://workers.cloudflare.com

For use cases that require additional processing of the configuration data or the inquiry form, serverless functions, as demonstrated in the example case, can be used. Unless a very large amount of requests is processed, these can also be very cost-effective, and the overview of the pricing per requests for three major serverless providers (Cloudflare Workers, AWS Lambda, Google Cloud Functions) as of April 2024 can be seen in Table 4.1.

In the example company described in this chapter, the only expense associated with the tool was the time spent preparing the content and deploying the application.

4.2.1.1 Return on Investment

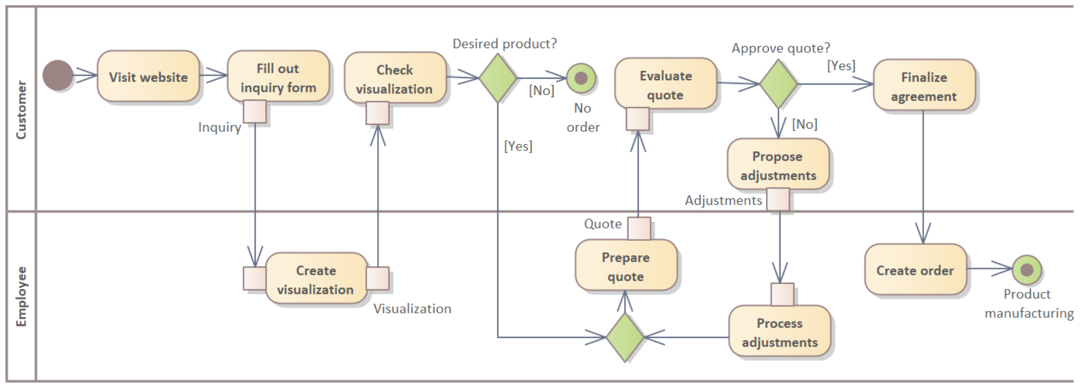
Consequently, the Return on Investment (ROI) for this solution is therefore projected to be hugely positive. While having minimal costs, the configurator not only acts as a sales assistance tool, allowing for detailed product visualization and customization, but also serves as a marketing tool that can enhance the company's market presence and appeal to a technically adequate consumer base.

4.2.2 Operational Impact

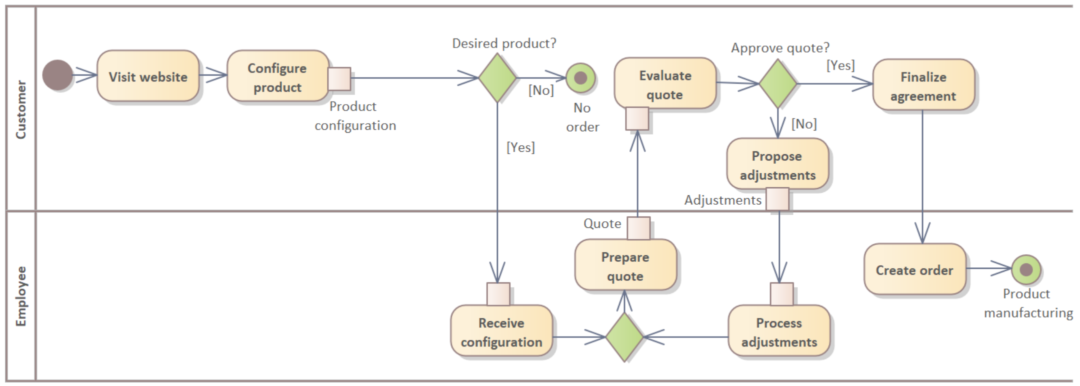
Evaluating the impact that the deployment of the solution brings to businesses and companies is a complex task. The toolkit is adaptable and product-agnostic, meaning it can be utilized by different companies across different industries or in different parts of a supply chain. Consequently, the impact on processes will differ in each scenario.

Therefore, this section will describe the change in processes after the tool's deployment in the example company examined in this chapter, as this should serve as an illustration of one of the common effects this tool can have.

This solution deployed in the example company impacts the inquiry and order process. This section will describe this process before and after implementation. The process involves two actors: the customer wanting to purchase a product and the company employee handling the customer's requests.



■ **Figure 4.1** Original order process as UML activity diagram



■ **Figure 4.2** New order process with the implemented solution as UML activity diagram

4.2.2.1 Original Process

The UML activity diagram of the original process is illustrated in Figure 4.1. The inquiry and order process before deployment of the solution looks as follows. Initially, when the customer wants to purchase a manufactured product, they visit the website of the company and fill out the inquiry form. Based on the information provided in the inquiry form, the employee creates a visualization, which is sent to the customer along with additional product customization informations. At this point, the customer decides whether this is the product they want or if they want to stop the process there. If the customer likes the product and options presented by the employee, an iterative process begins between the customer and the employee, where the employee tries to create an adequate quote for the product to the customer, while the customer sends back the proposed adjustments to the employee, who tries to process and incorporate them. When an agreement is reached, the order is placed, and the product goes into production.

4.2.2.2 New Process

After deploying the solution in the example company, as described in the previous sections, the inquiry and order process can change into the form illustrated in Figure 4.2. With this change, the customer visits the company's website and directly creates the configuration of the product they desire. The steps where the employee would need to present product visualizations and options are bypassed, as these are now handled by the tool. If the customer is satisfied with the configured product, they send an inquiry containing the product configuration, and the employee can process it and base the quote on this configuration. The process then follows in the same way as in the original scenario.

This demonstrates the power of the developed tool to simplify the initial step of the process, saving the employees time that would otherwise be spent on communicating with the customer. It also allows the customers to clearly express their preferences using the tool, reducing the friction of the inquiry process and potentially leading to more conversions into orders.

The tool also finds its application in marketing, where it can be used by the company to attract more potential customers.

Chapter 5
Testing

Unit testing, system testing, usability testing and gained insights.

Testing is an integral part of the software development process. This chapter provides an overview of the tests conducted during the development of this solution, as well as an evaluation and discussion of the testing results.

Software testing is essential for detecting malfunctions that may negatively affect the application's users and create further issues for the operator of the software. Testing is also necessary to confirm that the solution complies with the set specifications and helps to validate the design choices made. [65]

There are various types of software tests, which can be categorized in many different ways, such as automated versus manual, functional versus non-functional, black-box versus white-box, by their coverage (unit versus whole system) or by their scope (performance versus compatibility). The choice of tests is always context-dependent and needs to be aligned with the project goals and limitations. [66]

Testing 3D applications presents unique challenges that differentiate it from testing standard web applications, with main differences in user interactions. The developed solution enables users to move within a 3D space and interact with spatial objects. Interactions therefore occur in a 3D environment projected onto a 2D viewport, significantly expanding the range of possible interactions compared to traditional 2D applications. The potential interactions also vary depending on the product being configured in the application, as the solution aims to be product-agnostic, as well as further complicating matters with the configurator's open navigation mechanism.

Unlike 2D applications where layout and style are the focus, 3D applications require tests to confirm that objects appear correctly from various angles and under different conditions. This aspect is challenging, as it is less straightforward than verifying the DOM of a standard website, since in this scenario, the 3D preview is represented by a canvas element without a straightforward method for breakdown. In addition, loading 3D content is time and computationally consuming, which makes testing resource intensive. These complexities limit the applicability of certain types of tests, such as automated browser testing, which typically does not accommodate the nuances of working with 3D content.

Given these constraints and the goals of the solution, three distinct kinds of tests were performed during the development lifecycle of the application: unit tests, system tests, and usability tests. The tests performed are described in the following sections.

5.1 Unit Testing

Unit testing involves testing the smallest parts of the software, which are typically individual functions. The testing is performed in isolation from the rest of the system, with dependencies being mocked or stubbed to ensure that the tests are independent. These tests provide immediate feedback on the functionality of the written code and help detect bugs or regressions when the code undergoes changes during development. Since the resulting application is composed of these integrated parts, validating these parts helps with the validation of the whole software. [67]

```
describe("CatalogActions.getCatalog", () => {
  test(
    "returns the existing catalog "
    + "if already present in the store",
    async () => {
      const existingCatalog = generateMock(CatalogSchema);
      storeMock.catalog = existingCatalog;

      const catalog = await CatalogActions.getCatalog(
        "http://example.com/catalog",
        storeMock);

      expect(fetchCatalog).not.toHaveBeenCalled();
      expect(catalog).toBe(existingCatalog);
    }
  );
});
```

■ **Code listing 5.1** Example unit test used to validate store action within the solution

Due to the technologies chosen and the nature of the developed solution, the majority of the codebase is made up of code visualizing the products, written in markup language. This type of code is unsuitable for unit testing as it does not encompass any testable logic. However, parts of the solution that involve manipulation of data schemas in stores require complex logic and, therefore, these parts of the solution were subjected to unit testing.

The tests are stored in the `src/__tests__` directory. For the purpose of unit testing, the Jest framework[1] was used, which enables a streamlined definition and execution of the tests. These tests can be run by executing the following command: `npm run test`

Although tests should be isolated, they still operate with some data; therefore, these data schemas need to be mocked. Because data schemas were defined using the Zod framework (see Section 3.2), the zod-mock[2] along with faker packages were used to quickly generate fake data from the schemas to be used in these tests, which closely resemble the values that will be used in real-world scenarios. An example of a defined unit test can be seen in Code listing 5.1, which illustrates the generation of the mocked data, the calling of the tested function, and the verification of the results. Other unit tests are similar to the one illustrated.

These tests validate the logic within the application and were run when committing changes to confirm the integrity of the updates made to the code during the development process.

5.2 System Testing

System testing verifies that all integrated components and subsystems of the solution work as expected. This type of testing verifies that the application behaves as expected from the perspective of the user and that it meets the specified requirements. [25]

Due to the specifics of this solution outlined at the beginning of this chapter, this testing was not automated and was performed periodically manually during implementation. When defects were found, they were immediately addressed. Therefore, the evaluation of the fulfillment of the requirements, which should result from this testing, was done periodically and is described at the end of the implementation chapter (see Section 3.5).

To streamline this manual testing process, automatic deployment of the development environment was enabled using the GitLab CI/CD pipeline, which was triggered after each change to the codebase. To assess various different functionalities of the application, several different sample products, such as computer or shelves configurations, were created to enable the system testing process.

To ensure the compatibility and responsiveness of the application across different browsers and operating systems, the LambdaTest platform[3] was utilized. This platform enables the testing of web applications on thousands of combinations of major browsers and operating systems. [68]

[1]`https://jestjs.io`
[2]`https://www.npmjs.com/package/@anatine/zod-mock`
[3]`https://www.lambdatest.com/`

Therefore, the application was tested on a relevant sample of browsers. The solution is designed to be compatible with all major browsers (Chrome, Firefox, Safari, Opera, Edge) with versions released from the year 2022 onward.

5.3 Usability Testing

Usability testing evaluates the functionality of the web application. In contrast with the system testing described in the previous section, usability tests employ real users. They are carried out by a facilitator, along with a selected sample of users, who perform tasks representative of actual usage situations. The facilitator observes the users as they complete the tasks, noting their interactions with the system and their reactions. This allows to measure effectiveness, efficiency, and satisfaction of using the software. Usability testing is important, as it validates the proposed design from a fresh point of view, as the perception of the developed solution is different between the developer, who already knows every detail of the application, and the user, who has to learn how to work with the application. [69]

Various categories of usability tests can be performed to assess different aspects of usability. Quantitative testing focuses on gathering numerical data about user experience, while qualitative testing collects observations and subjective feedback. In moderated testing, the user is led by the facilitator, compared to unmoderated testing, where the user is acting independently. Testing can be done remotely or in person, depending on available resources. [70]

5.3.1 Test Plan

Before the actual testing process takes place, it is necessary to establish what is tested, in what manner, who are the test participants, and how is the testing conducted and evaluated. The following section discusses these aspects of the solution's usability testing.

In this thesis, usability testing is performed only on the configurator application. As discussed in the introduction of this thesis, the enjoyment of the configuration process itself directly influences the perceived value of the configured product [4], therefore, it is imperative to ensure that the customer-centric configurator application is highly efficient and provides a great user experience in order to maximize the satisfaction of customers using the tool to configure their products.

Although usability testing could also be extended to the administrator part of the toolkit, it is important to note that the administrator application will be used only by administrators, which are only a few compared to the users of the configurator application. Administrators are also expected to have better technical knowledge than ordinary customers. The administrator part of the toolkit is conceived to be used infrequently, typically during the initial setup of the tool or for occasional updates of the product catalog. Given these factors, the configurator part of the toolkit must comply with usability standards higher than those of the administrator part; therefore, the available testing resources are better allocated entirely to the configurator application.

Usability testing was performed in person and in locations that are natural to the participants, either at their workplaces or at their homes. Since the solution requires no installation, the test participant's personal device was used to simulate real-world conditions and keep the participants comfortable.

The comfort of the test participant is important during the testing process, as anxious users can skew the results by not reacting the same way as they would naturally when using the tool outside of a testing context. [71]

Based on analysis of performed usability testing, research has revealed that, on average, five test participants are enough to reveal around 85% of usability flaws, with additional participants providing diminishing returns in terms of unique feedback [72]. For this reason, the usability test was performed with five different users to efficiently gather as much comprehensive information as possible without overly intensive testing.

During testing, standard approaches were followed. These include recording the sessions to enable for more objective processing afterward. The testing process was clearly explained, making it known to the participant that the solution is being tested, not the user, and therefore they are free to express themselves and will not be judged. In addition, the facilitator aimed to conduct the test in a way that does not lead or influence the test participant by helping in any way or asking suggestive questions during the testing process. [70]

5.3.1.1 Evaluation Methods

Usability testing was performed using both qualitative and quantitative methods.

To gather important context before the testing process, participants were, along with their general profile, asked about the following information:

1. Does the participant have any previous experience with online product configuration, has the participant ever used similar tool to preview or order a product?
2. Does the participant have experience with 3D computer programs?
3. When ordering custom products, does the participant prefer personal contact, such as phone or email communication, or would they rather use an online configurator?

This information is relevant, as the 3D controls employed in the developed solution are common; therefore, previous experience could impact the results of the usability test. In addition, the sentiment regarding the preferred shopping process could also influence the perceived usability of the tool.

During the process, the actions of the participants were observed and the think-aloud method was used.

With the think-aloud protocol, participants are encouraged to reveal their thoughts during the process and to think out loud when using the application. This means that along with the actions taken, the information about why they have taken it is also immediately captured. [70]

After the test, a qualitative evaluation was done by conducting an interview with a general discussion about the solution, including the following questions:

1. What difficulties did the participant encounter when using the tool? Was there anything particularly frustrating?
2. What three improvements or features would the participant like to see?
3. Did any part of the tool feel unnecessary or redundant?
4. Is the product representation clear enough to understand what was configured?
5. Would the participant use this tool in a real scenario to inquire configured products?
6. What was the initial feeling of the participant about the tool?

For quantitative evaluation, the System Usability Scale (SUS) questionnaire was used.

The SUS is an industry standard for quickly evaluating the usability of a system. It consists of 10 questions, each with a numerical response on a scale from one to five, one representing "strongly disagree" and five being "strongly agree". The questions are presented in Table 5.1. [73]

A score is then calculated from the SUS questionnaire as follows:

- The response values for odd-numbered questions are subtracted by one.
- The response values for even-numbered questions are subtracted from five.

The adjusted responses are then added together and multiplied by 2.5, giving a score between 0 and 100. This is calculated for each participant. To get a single value that rates the entire system, the average of these scores is taken. [74]

Research of 500 usability studies has shown that the average SUS score is 68. Therefore, a score above 68 suggests that the solution is better than the average system in terms of usability. However, it should be noted that this is a comparison with different types of systems, not necessarily with the same functionality. The 90th percentile is an SUS score of about 80.3. [74]

The time to complete the tasks was not measured, as product configuration is a creative process where speed does not necessarily mean better performance.

1.	I think that I would like to use this system frequently
2.	I found the system unnecessarily complex
3.	I thought the system was easy to use
4.	I think that I would need the support of a technical person to be able to use this system
5.	I found the various functions in this system were well integrated
6.	I thought there was too much inconsistency in this system
7.	I would imagine that most people would learn to use this system very quickly
8.	I found the system very cumbersome to use
9.	I felt very confident using the system
10.	I needed to learn a lot of things before I could get going with this system

■ **Table 5.1** System Usability Scale questionnaire
Source: [73]

5.3.1.2 Scenario

For usability testing purposes, a mock product, a modular kitchen countertop, was prepared for configuration within the tool, with configurable components such as drawers, sinks, cabinets, corners and smooth edges.

Consequently, a single test case was used that involved the configuration of the defined kitchen product. This test case was broken down into several subtasks that participants were instructed to complete using the configurator application.

In the scenario presented to the users, the background context involved them renovating their homes and being in need of a new remodeled kitchen. They found a company specialized in custom kitchen furniture, which utilized a website where they used the developed solution to handle inquiries. The user has visited the configurator, at which point the usability testing has started. The subtasks were as follows:

1. Create a configuration of kitchen modules in “L” shape. The left part of the kitchen should include a drawer and sink, and the right part of the kitchen should contain two cabinets and another drawer.

2. Add a smooth edges component to the edge modules.

3. Set the color of the countertop of all the modules to “dark” and the wooden part of the modules to the color “cherry”. The color of the sink component should be set to “copper”, and the faucet to “gold” color.

4. Review the created configuration and fill out and send an inquiry form with contact details.

These product features and subtasks were selected to ensure that the majority of functionalities of the configurator application were tested by the users in a single, comprehensive test case, which represented the expected regular use.

5.3.2 Testing Process

The following section describes the selection of participants for the usability test and provides a summary of the transcription from the testing process itself.

5.3.2.1 Participants

As mentioned in the previous section, five participants were selected, which should be sufficient to uncover most usability issues. The participants were deliberately chosen to be heterogeneous, representing a range of different age groups, to ensure that the testing provided as much information as possible.

The participants, along with the contextual information gathered from them and the devices they used, are as follows:

Participant A: 18 years old, male

- **Experience:** Previous experience with 2D car configurator, also relevant experience with 3D modeling and extensive 3D gaming.
- **Device:** Desktop PC with Windows.

Participant B: 22 years old, female

- **Experience:** Previous experience with 3D furniture configurator, also relevant experience with casual 3D gaming.
- **Device:** Mobile device with iOS.

Participant C: 46 years old, female

- **Experience:** Previous experience with 3D furniture configurator, no other relevant experience.
- **Device:** Desktop PC with MacOS (with an Apple Magic Mouse).

Participant D: 52 years old, male

- **Experience:** Previous experience with 3D car configurator, also relevant experience with 3D visualization software.
- **Device:** Laptop with MacOS (with a standard external mouse).

Participant E: 62 years old, female

- **Experience:** No previous experience with configurators or other relevant experiences.
- **Device:** Laptop with Windows (with a standard external mouse).

All participants expressed a positive sentiment towards using a configurator tool instead of a direct inquiry process.

5.3.2.2 Execution and Observations

The testing process began by preparing the tested application on the device and ensuring that there would be no interruptions during the testing.

The usability testing process, its purpose, and principles were then explained to the participants along with the three stages that it consists of: pre-test questions, the test itself, and post-test questions.

Questions were asked to establish context about the participant's background, experience, and sentiment, as detailed in the previous sections. The responses can be found in the previous section along with other details about the participants.

The test of the application itself then began. The participants were reminded that it was the application being tested, not the user; therefore, they would not be judged and that any issues they may face are problems of the application's usability. Participants were also reminded that they would not be helped during the process but only observed, and were asked to think aloud and share their thoughts to help with this observation. The prepared scenario and the tasks they should complete were explained and presented to them on a sheet of paper for further easy reference.

All participants immediately understood what to do and chose the initial component of the kitchen. The controls in the 3D space, along with the buttons for the addition of components, were also understood quickly by everyone. Therefore, the addition of the first components of the product was not problematic for anyone.

However, as the configured product grew more complex, the addition of more components became an issue for everyone except Participant A. The problem was caused by an unclear scrolling pattern in the component addition menu. The menu pops up from the bottom of the screen, with horizontally laid tiles representing the mountable components. If there are more mountable components than would fit in this menu, horizontal scrolling is utilized. Due to the disappearance of the scroll bar and soft shading of the background, it was not clear to the participants that there could be more components accessible by scrolling horizontally. Although every participant has eventually figured it out, it has caused problems and frustrations, which, especially in the case of Participant C, lasted several minutes. Furthermore, Participant D has also suggested that a categorization of the available components on this panel would be beneficial for better orientation.

Another observed pain point, which the majority of participants have also reported afterwards, was aggressive camera zooming. Whenever a component is selected, the camera tries to center the chosen component on screen. Almost all participants have reported that this movement is either too jarring and fast or that the zoom effect is too strong and that the 3D object should not be zoomed so much.

Participant C, who used an Apple Magic Mouse with an unconventional button layout and gestures (see Figure 5.1), experienced particular difficulties. When trying to compensate for unwanted camera centering, the participant accidentally performed a gesture on the mouse to navigate away from the application, restarting the whole configuration.

Figure 5.1 Apple Magic Mouse
Source: Apple [75]

In addition to the selectable components in the 3D view, as a supporting element, there is also a small button symbolizing the component that also allows users to select it. This button is positioned at the point where the component is mounted. This has caused small issues for Participant C and Participant E. The post-test discussion revealed that the problem was caused by two things: the base component does not have a mounting point, therefore, the application does not have a button for this component, and at the same time the position of the button on the mounting point caused confusion as it was unclear to which component the button belongs to. The participants implied that it would be more intuitive if the button was placed in the center of the component.

Participants C and E also encountered small issues with the remove component button, which uses a hold-to-confirm mechanism instead of the traditional confirmation pop-up. The unfamiliarity with this mechanism has led both participants to require three attempts to perform this action.

Participant B expected that the review of components in the confirmation screen would be interactive, allowing for changes to the colors of the materials instead of merely reviewing them.

After overcoming these challenges, the participants quickly understood and completed the rest of the tasks without any further obvious problems. The participants were then asked to complete the SUS questionnaire on the second side of the instructions sheet, followed by an interview where the previously detailed questions were asked.

In the post-test interview, a common wish among all participants except Participant D was that the material color changes would apply to all components made of the same material, rather than having to adjust each one individually. Two participants also mentioned that the outline of the selected components was too wide and strong, obscuring the customized color. Participant B has also expressed the desire to have the ability to change the environments in which the product is visualized.

Participants deemed no part of the tool redundant, except Participant D, who considered the configuration review and confirmation screen unnecessary in the current form and suggested that a screenshot of the 3D preview of the configuration should also be included there.

When asked about their initial feeling, three participants expressed that they wanted a guide or tutorial to be presented on the initial screen, and two participants mentioned that having preset configurations with prearranged components would be beneficial.

All of the participants regarded the 3D product representation to be clear and understandable and all expressed that they would use this application in a real scenario.

5.3.3 Test Results

The following section provides evaluation results and a summary of insights gained from the testing process, as detailed in the previous section.

The general sentiment regarding the usability of the tool was positive among all participants.

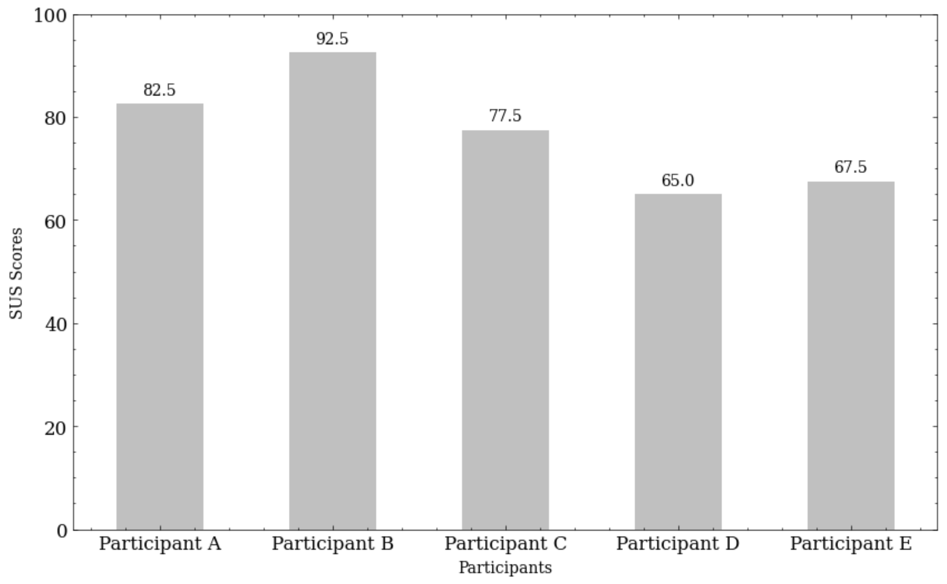


Figure 5.2 System Usability Scale scores by participant

The average SUS score for all participants was 77 which is considered good and above average when compared to other systems. The lowest SUS score was 65, recorded by Participant D, and the highest was 92.5, recorded by Participant B. The scores for all participants are plotted on the chart shown in Figure 5.2.

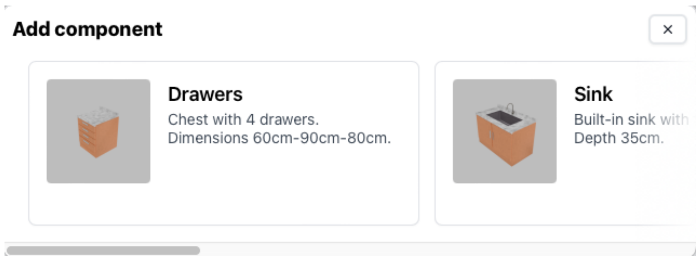
A trend appears to indicate that the tool is more suitable for younger users. However, given the small sample size, this does not provide enough evidence to draw statistically significant conclusions in this regard.

5.3.3.1 Insights

The following section summarizes the issues identified during usability testing along with potential improvements, describes how fixes could be implemented, and reports the current status of these implementations. In addition, the severity of each issue is estimated based on the difficulties observed during usability testing. The estimated severity was classified as high, medium, or low. The priority for implementing fixes was based on the estimated severity.

I1: Unclear scrolling pattern in component addition menu
Horizontal scrolling in the menu is not apparent (see Figure 5.3), leading users to believe that there are fewer available components than there really are.

- **Severity:** High
- **Proposed fix:** Arrow buttons should be added as an additional apparent scrolling mechanism
- **Implemented:** Yes, see Figure 5.4

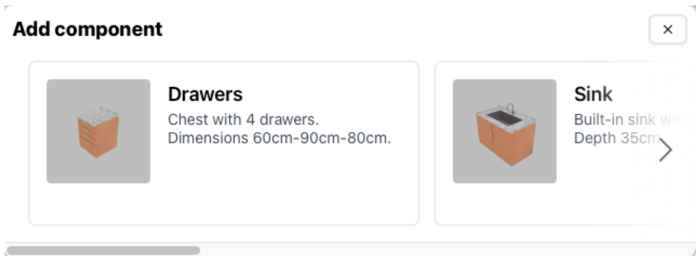

Add component

×

Drawers
Chest with 4 drawers.
Dimensions 60cm-90cm-80cm.

Sink
Built-in sink with
Depth 35cm.

Figure 5.3 Component addition menu before the implemented fix


Add component

×

Drawers
Chest with 4 drawers.
Dimensions 60cm-90cm-80cm.

Sink
Built-in sink wi
Depth 35cm

Figure 5.4 Component addition menu with implemented arrows for scrolling

I2: Jarring camera movement

The camera moves too abruptly and zooms in too much on component selection, causing discomfort and confusion among users.

- **Severity:** High
- **Proposed fix:** The camera speed should be slowed down and limited to rotation only, avoiding zoom on component selection
- **Implemented:** Yes

I3: Incorrect position of the component selection button

The button supporting the selection of components is positioned at the point the component is mounted at, creating confusion about which component the button is associated with. The button is also missing on the base component.

- **Severity:** Medium
- **Proposed fix:** The button representing the component should be centered within the 3D model and also added to the base component
- **Implemented:** Yes

I4: Excessive outline of the selected component

The outline of the selected component within the 3D preview can be too wide, obscuring the customized color of the component.

- **Severity:** Medium
- **Proposed fix:** The width of the outline should be reduced or made transparent
- **Implemented:** Yes

I5: Lack of guidance

The tool does not provide users with any information about the controls.

- **Severity:** Medium
- **Proposed fix:** A panel detailing the controls should be presented when first accessing the application and also be accessible by a button at any time
- **Implemented:** No

I6: Separated material changes

Changing the color of materials only affects a single component, even if there are other components with the same materials in the configuration. Therefore, updating the same material across all components requires individually adjusting each one.

- **Severity:** Medium
- **Proposed fix:** The data scheme for material specifications should be revised to not be part of the components but to be shared between them, alternatively, changing the color of a material could automatically update all components with the same material identification
- **Implemented:** No

I7: Missing preset configurations

The configurator does not provide users with precreated configurations on which the user could build upon.

- **Severity:** Medium
- **Proposed fix:** Administrator should be able to create and offer product configurations which the user can use to derive their own configuration
- **Implemented:** No

I8: Unintuitive hold-to-confirm mechanism

The confirmation action used on the delete button, which requires holding the button for a while, is unintuitive.

- **Severity:** Low
- **Proposed fix:** The hold-to-confirm mechanism should be replaced with standard confirmation popup
- **Implemented:** No

I9: Confirmation screen does not allow changes

The confirmation screen only offers an overview, and to make changes to the colors of materials, it is necessary to return to the configurator screen. In addition, the confirmation screen could provide more information.

- **Severity:** Low
- **Proposed fix:** Changing colors of materials should be enabled directly from the confirmation screen, and the screen should present a screenshot of the preview of the 3D configuration
- **Implemented:** No

I10: Lack of categorization in the component addition menu

Categorization of components in the addition menu would improve user orientation and ease of use.

- **Severity:** Low

- **Proposed fix:** A new category data scheme should be implemented to encompass different component specifications, and the menu should group the components based on their respective categories
- **Implemented:** No

I11: Blank environment in the 3D preview

The 3D preview features a blank background with color set by the administrator, which can appear bland.

- **Severity:** Low
- **Proposed fix:** Various 3D backgrounds should be introduced to allow visualization of the configured products in different environments
- **Implemented:** No

Since only the most critical usability improvements were implemented, the remaining unimplemented usability enhancements could be the subject of further development of the tool and complement Subsection 3.5.1 on future improvements.

Chapter 6
Conclusion

The objective of this bachelor's thesis was to create an application for online 3D configuration of modular products.

To accomplish this goal, an analysis of existing solutions was performed, examining the perspective of both customers and the businesses that operate such tools. Based on this, requirements for the solution implemented in this thesis were created, such that the created application provides businesses with an innovative solution for product configuration.

Consequently, a responsive web application was designed to support the configuration of a variety of modular products in a 3D environment. The configurator features open navigation, allowing users to customize modular products by adding or removing specified components at fixed points, and to customize their properties such as the color of the materials. Additionally, advanced features such as collision detection were included to ensure that configurations are realistic and feasible. Further processing of the user-created configurations has been made available by offering an integration using an API call to other systems or by presenting an inquiry form.

From a business perspective, the subsequent goals were for the application to be flexible, lightweight, and easy to maintain, targeting smaller companies in need of a cost-effective solution. The developed tool is a front-end-only solution, set up by static files. To allow customers to configure their products, businesses can deploy the developed solution directly on their webservers. The tool as a whole has been separated into two separate applications: the user-facing configurator and an administrator tool allowing the business to define their configurable products.

In the design chapter, technologies were chosen and wireframes of the user interfaces were crafted.

The implementation chapter detailed the data schemas and the solutions to several interesting challenges encountered during the implementation of the designed application.

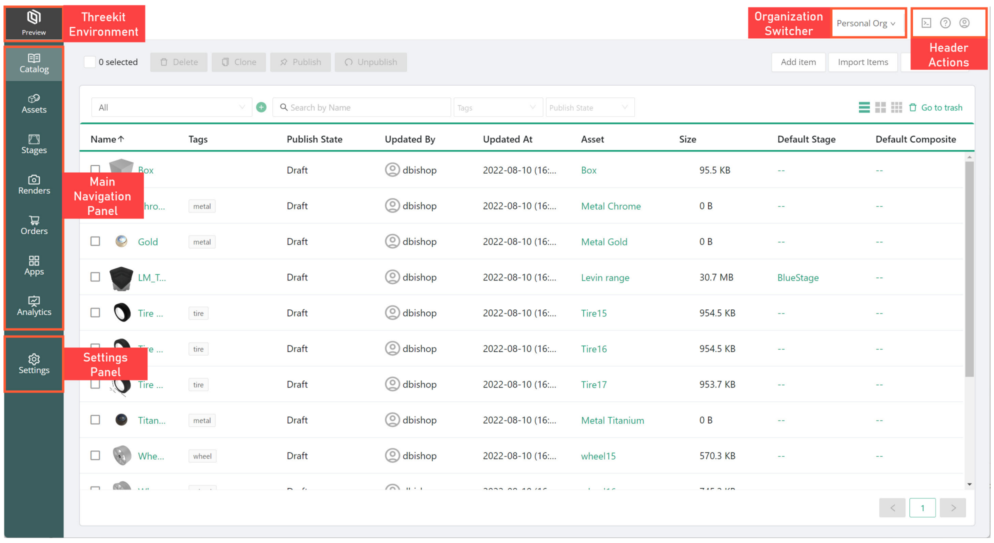
In the deployment chapter, the deployment process was described within the context of an example business, highlighting the potential changes to the business processes enabled by this solution.

Furthermore, the testing performed during the development of the application was discussed, including unit testing, system testing, and usability testing. Usability testing was conducted near the end of the development cycle to validate the functionality and user experience of the application, and the most severe revealed usability issues were fixed.

Future improvements to this solution could include the creation of an accompanying back-end solution, integrations with existing e-commerce platforms, or the implementation of a rule evaluation engine. Possible future development directions are discussed in detail in Subsection 3.5.1, and insights from usability testing for user experience improvements are also provided in Subsubsection 5.3.3.1.

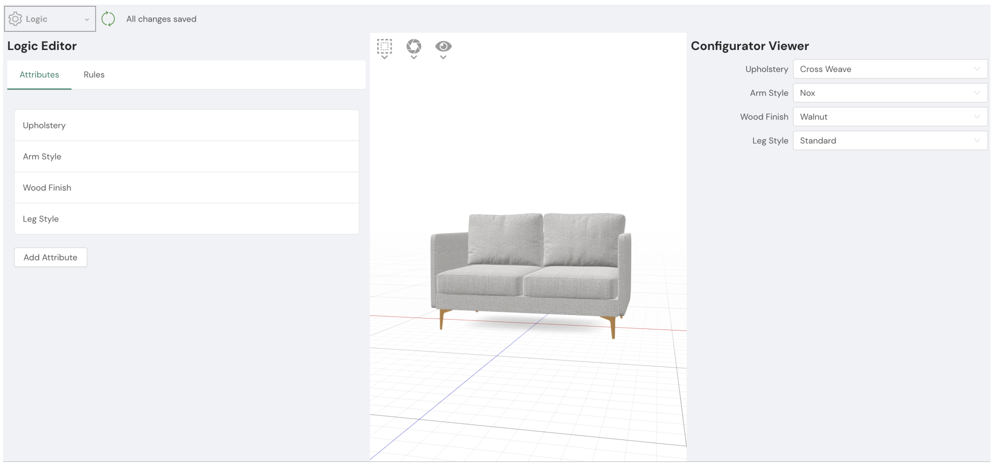
The developed solution is freely available for businesses to use, enabling them to introduce modular product configuration on their websites effectively.

Appendix A
Additional Visuals



Name↑	Tags	Publish State	Updated By	Updated At	Asset	Size	Default Stage	Default Composite
Box		Draft	dbishop	2022-08-10 (16:...	Box	95.5 KB	--	--
hro...	metal	Draft	dbishop	2022-08-10 (16:...	Metal Chrome	0 B	--	--
Gold	metal	Draft	dbishop	2022-08-10 (16:...	Metal Gold	0 B	--	--
LM_T...		Draft	dbishop	2022-08-10 (16:...	Levin range	30.7 MB	BlueStage	--
Tire ...	tire	Draft	dbishop	2022-08-10 (16:...	Tire15	954.5 KB	--	--
e ...	tire	Draft	dbishop	2022-08-10 (16:...	Tire16	954.5 KB	--	--
Tire ...	tire	Draft	dbishop	2022-08-10 (16:...	Tire17	953.7 KB	--	--
Titan...	metal	Draft	dbishop	2022-08-10 (16:...	Metal Titanium	0 B	--	--
Whe...	wheel	Draft	dbishop	2022-08-10 (16:...	wheel15	570.3 KB	--	--

■ **Figure A.1** Threekit's Platform's landing page
Source: Threekit Platform Documentation [20]



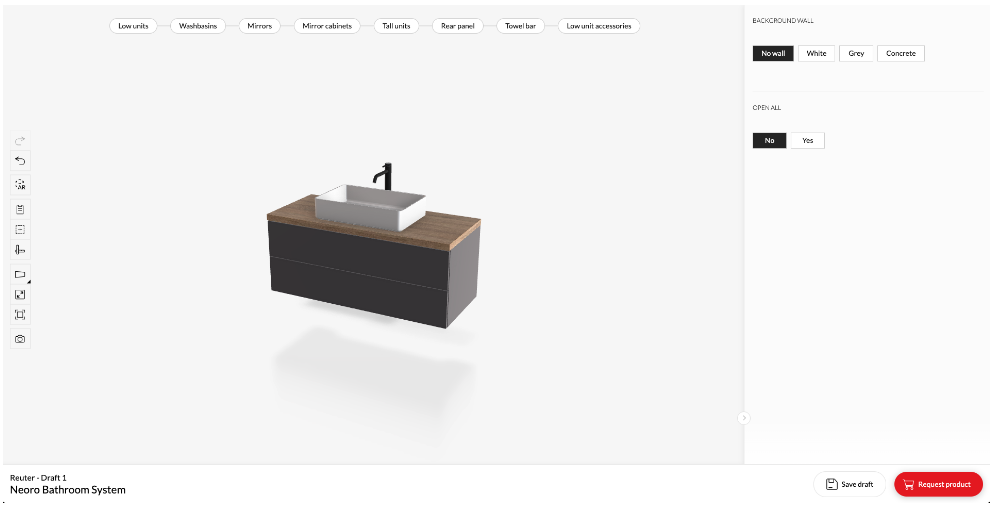
Logic Editor

Upholstery
Arm Style
Wood Finish
Leg Style

Configurator Viewer

Upholstery	Cross Weave
Arm Style	Nox
Wood Finish	Walnut
Leg Style	Standard

■ **Figure A.2** Threekit's Platform's editor
Source: Threekit Platform Documentation [20]



BACKGROUND WALL

[✓ No wall] White Grey Concrete

OPEN ALL

[✓ No] Yes

Reuter - Draft 1
Neoro Bathroom System

■ **Figure A.3** Screenshot of Roomle's Rubens example
Source: Roomle Demos [23]

Appendix B

Additional Code Listings

```
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

const scene = new THREE.Scene();

const geometry = new THREE.BoxGeometry(5, 5, 5);
const material = new THREE.MeshBasicMaterial({color: 0xff0000});
const mesh = new THREE.Mesh(geometry, material);
scene.add(mesh);

const camera = new THREE.PerspectiveCamera(
  75,
  window.innerWidth / window.innerHeight,
  0.1,
  1000
);
camera.position.set(10, 10, 10);
camera.lookAt(mesh.position);

renderer.render(scene, camera);
```

■ **Code listing B.1** Creating and displaying a 3D red cube with Three.js

```
const Component = () => {
    return (
        <Canvas camera={{position: [10, 10, 10]}}>
            <mesh>
                <meshBasicMaterial color="red" />
                <boxGeometry args={[5, 5, 5]} />
            </mesh>
        </Canvas>
    )
}
```

■ **Code listing B.2** Creating a 3D red cube as a React component with R3F

```
import { z } from 'zod';

const customSchema = z.number();

type CustomType = z.infer<typeof customSchema>;
```

■ **Code listing B.3** Conversion from Zod schema to TypeScript type
Source: Adapted from [56]

Bibliography

1. FULKERSON, Bill; SHANK, Michael. The New Economy Electronic Commerce, and the Rise of Mass Customization. In: *Handbook on Electronic Commerce*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 411–430. ISBN 978-3-540-67344-6.
2. FRANKE, Nikolaus; PILLER, Frank. Configuration toolkits for mass customization. *Arbeitsberichte des Lehrstuhls für Allgemeine und Industrielle Betriebswirtschaftslehre der Technischen Universität München*. 2002, vol. 33, no. 4, p. 28.
3. SCHREIER, Martin. The value increment of mass-customized products: an empirical assessment. In: [online]. 2006, vol. 5, pp. 317–327. No. 4. Available from DOI: `10.1002/cb.183`. Accessed 2024-01-25.
4. FRANKE, Nikolaus; SCHREIER, Martin. Why Customers Value Self-Designed Products: The Importance of Process Effort and Enjoyment*. In: [online]. 2010, vol. 27, pp. 1020–1031. No. 7. Available from DOI: `10.1111/j.1540-5885.2010.00768.x`. Accessed 2024-01-25.
5. COZZI, Patrick. *WebGL Insights*. CRC Press, 2015. ISBN 978-1498716079.
6. ZHAO, Huiwen; MCLOUGHLIN, Leigh; ADZHIEV, Valery; PASKO, Alexander. An Evaluation Model for Web-based 3D Mass Customization Toolkit Design. In: *Customization 4.0*. Cham: Springer International Publishing, 2018, pp. 375–390. ISBN 978-3-319-77555-5.
7. HERMANS, Guido. A Model for Evaluating the Solution Space of Mass Customization Toolkits. *International Journal of Industrial Engineering and Management* [online]. 2012. Available from DOI: `10.24867/ijiem-2012-4-125`. Accessed 2024-01-25.
8. CYLEDGE MEDIA. *Configurator Database* [online]. Vienna, 2018. Available also from: `https://www.configurator-database.com/`. Accessed 2024-01-25.

9. BLAZEK, Paul. Creating Customization Experiences: The Evolution of Product Configurators. In: *Mass Customization and Customer Centricity: In Honor of the Contributions of Cipriano Forza* [online]. Cham: Springer International Publishing, 2023, pp. 179–209. ISBN 978-3-031-09782-9. Available from DOI: `10.1007/978-3-031-09782-9_7`. Accessed 2024-01-25.
10. STATISTA RESEARCH DEPARTMENT. *IKEA - Statistics & Facts* [online]. Statista, 2024. Available also from: `https://www.statista.com/topics/1961/ikea`. Accessed 2024-01-30.
11. IKEA. *Design your PAX storage* [online]. [N.d.]. Available also from: `https://www.ikea.com/addon-app/storageone/pax/web/latest/cz/en/`. Accessed 2024-01-30.
12. FINK, Gil; FLATOW, Ido. Introducing Single Page Applications. In: *Pro Single Page Application Development: Using Backbone.js and ASP.NET*. Berkeley, CA: Apress, 2014, pp. 3–13. ISBN 978-1-4302-6673-0. Available from DOI: `10.1007/978-1-4302-6674-7_1`.
13. MUUTO. *Our Story* [online]. ©2023. Available also from: `https://www.muuto.com/content/about/our-story-a-space-that-just-feels-right/`. Accessed 2024-01-30.
14. MUUTO. *Product Planner* [online]. [N.d.]. Available also from: `https://planner.muuto.com/`. Accessed 2024-01-30.
15. JACKSON, Dean. Viewing Augmented Reality Assets in Safari for iOS. In: *WebKit* [online]. 2018. Available also from: `https://www.webkit.org/blog/8421/viewing-augmented-reality-assets-in-safari-for-ios/`. Accessed 2024-01-30.
16. LD SEATING. *About us* [online]. ©2024. Available also from: `https://www.ldseating.com/en/about-us`. Accessed 2024-01-30.
17. LD SEATING. *Nido* [online]. [N.d.]. Available also from: `https://nido.ldseating.com/en/configurator`. Accessed 2024-01-30.
18. THREEKIT INC. *About us* [online]. ©2023. Available also from: `https://www.threekit.com/en/about-us`. Accessed 2024-01-31.
19. THREEKIT INC. *The Enterprise Platform for Visual Commerce* [online]. ©2023. Available also from: `https://www.threekit.com/en/platform-overview`. Accessed 2024-01-31.
20. THREEKIT INC. *Platform Documentation* [online]. 2024. Available also from: `https://community.threekit.com/v/platform-documentation`. Accessed 2024-01-31.
21. ROOMLE GMBH. *Make your furniture a better digital experience* [online]. ©2023. Available also from: `https://www.roomle.com/en/about`. Accessed 2024-01-31.

22. ROOMLE GMBH. *Documentation* [online]. ©2020. Available also from: `https://docs.roomle.com/`. Accessed 2024-02-01.

23. ROOMLE GMBH. *Full-logic Configurator* [online]. ©2023. Available also from: `https://www.roomle.com/en/demos/full-logic-configurator`. Accessed 2024-02-01.

24. AURUM, Aybüke; WOHLIN, Claes. Requirements Engineering: Setting the Context. In: *Engineering and Managing Software Requirements*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1–15. ISBN 978-3-540-28244-0. Available from DOI: `10.1007/3-540-28244-0_1`. Accessed 2024-02-04.

25. STEPHENS, Rod. *Beginning Software Engineering*. 2nd ed. Hoboken, NJ, USA: Wiley, 2022. ISBN 978-1-119-90170-9.

26. PARISI, Tony. *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2014. ISBN 978-1-449-36296-6.

27. PARISI, Tony. *WebGL: Up and Running*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2012. ISBN 978-1-4493-2357-8.

28. WORLD WIDE WEB CONSORTIUM. *WebGPU* [online]. 2024-02. W3C Working Draft. World Wide Web Consortium (W3C). Available also from: `https://www.w3.org/TR/2024/WD-webgpu-20240202/`. Accessed 2024-02-05.

29. CATUHE, David et al. *Babylon.js* [online]. 2023. Available also from: `https://github.com/BabylonJS/Babylon.js`. Accessed 2024-02-05.

30. CABELLO, Ricardo et al. *Three.js - JavaScript 3D library* [online]. 2023. Available also from: `https://threejs.org/`. Accessed 2024-02-05.

31. GIMENO, Alberto. The deepest reason why modern JavaScript frameworks exist. In: *DailyJS* [online]. 2018. Available also from: `https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445`. Accessed 2024-02-06.

32. PEKARSKY, Max. *Does your web app need a front-end framework?* [online]. 2020. Available also from: `https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/`. Accessed 2024-02-06.

33. HENSCHEL, Paul et al. *React Three Fiber Documentation* [online]. 2024. Available also from: `https://docs.pmnd.rs/react-three-fiber`. Accessed 2024-02-06.

34. HENSCHEL, Paul et al. *useful helpers for react-three-fiber* [online]. 2024. Available also from: `https://github.com/pmndrs/drei`. Accessed 2024-02-06.

35. BANKS, Alex; PORCELLO, Eve. *Learning React: Modern Patterns for Developing React Apps*. 2nd. Sebastopol, CA, USA: O'Reilly Media, Inc., 2020. ISBN 978-1-492-05172-5.

36. EZE, Peter Ekene. *Next.js vs. Gatsby: Comparing React frameworks* [online]. 2023. Available also from: `https://blog.logrocket.com/next-js-vs-gatsby-comparing-react-frameworks/`. Accessed 2024-02-06.

37. SAID, Mostafa. *Vite vs. Webpack: A Head-to-Head Comparison* [online]. kinsta, 2023. Available also from: `https://kinsta.com/blog/vite-vs-webpack/`. Accessed 2024-02-06.

38. ABBA, Ihechikara. *How to Use Tailwind CSS to Rapidly Develop Snazzy Websites* [online]. kinsta, 2023. Available also from: `https://kinsta.com/blog/tailwind-css/`. Accessed 2024-02-06.

39. MICROSOFT. *Why TypeScript* [online]. 2023. Available also from: `https://www.typescriptlang.org/why-create-typescript`. Accessed 2024-02-06.

40. GANATRA, Sagar. *React Router Quick Start Guide*. Packt Publishing, 2018. ISBN 978-1789532555.

41. KRUKOWSKI, Ilya. *Go Global with React and i18next: A Comprehensive Tutorial for Internationalizing Your React App* [online]. lokalise, 2023. Available also from: `https://lokalise.com/blog/how-to-internationalize-react-application-using-i18next/`. Accessed 2024-04-08.

42. CEDDIA, Dave. *React State Management Libraries and How to Choose* [online]. 2021. Available also from: `https://daveceddia.com/react-state-management/`. Accessed 2024-04-08.

43. ADEPOJU, Opeyemi. *State Management In React With Valtio* [online]. OpenReplay, 2023. Available also from: `https://blog.openreplay.com/state-management-in-react-with-valtio/`. Accessed 2024-04-08.

44. BHIMANI, Kesar. *Zod and React: A Perfect Match for Robust Validation* [online]. DhiWise, 2023. Available also from: `https://www.dhiwise.com/post/zod-and-react-a-perfect-match-for-robust-validation`. Accessed 2024-04-08.

45. PONUTHORAI, Prem Kumar; LOELIGER, Jon. *Version Control with Git*. 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2022. ISBN 9781492091196.

46. GITLAB. *Gitlab Docs* [online]. 2024. Available also from: `https://docs.gitlab.com`. Accessed 2024-04-07.

47. PETRUNGARO, Damiano et al. *Conventional Commits 1.0.0* [online]. 2022. Available also from: `https://www.conventionalcommits.org/en/v1.0.0/`. Accessed 2024-04-07.

48. TYPICODE. *Husky Introduction* [online]. 2024. Available also from: `https://typicode.github.io/husky/`. Accessed 2024-04-07.

49. ABRAMOWSKI, Nicole. *What is NPM? The Complete Beginner's Guide* [online]. CareerFoundry, 2022. Available also from: `https://careerfoundry.com/en/blog/web-development/what-is-npm/`. Accessed 2024-04-07.

50. GUPTA, Shivam. *ESLint: What, Why, When, How* [online]. dev.to, 2021. Available also from: `https://dev.to/shivambmgupta/eslint-what-why-when-how-5f1d`. Accessed 2024-04-07.

51. WOJTASIŃSKI, Paweł. Why the Hell Do People Confuse Prettier With Eslint? In: *Hackernoon* [online]. 2023. Available also from: `https://hackernoon.com/why-the-hell-do-people-confuse-prettier-with-eslint`. Accessed 2024-04-07.

52. HANSEN, Torben; SCHEER, Chris J.; LOOS, Peter. Product Configurators in Electronic Commerce - Extension of the Configurator Concept towards Customer Recommendation. In: *MCPC* [online]. 2003. Available also from: `https://api.semanticscholar.org/CorpusID:6859338`. Accessed 2024-01-26.

53. WLASCHIN, Scott. *Domain Modeling Made Functional*. Pragmatic Bookshelf, 2018. ISBN 978-1680502541.

54. LEITNER, Gerhard; FELFERNIG, Alexander; BLAZEK, Paul; REINFRANK, Florian; NINAUS, Gerald. User Interfaces for Configuration Environments. In: *Knowledge-Based Configuration: From Research to Business Cases*. Boston: Morgan Kaufmann, 2014, pp. 89–106. ISBN 978-0-12-415817-7. Available from DOI: `10.1016/B978-0-12-415817-7.00008-6`.

55. LORANGER, Hoa; SCHADE, Amy; NIELSEN, Jakob. *Website Tools and Applications with Flash* [online]. 2013. report. Nielsen Norman Group. Available also from: `https://www.nngroup.com/reports/website-tools-and-applications-flash/`. Accessed 2024-04-29.

56. WYCLIFFE, Maina. Using Zod Schemas as Source of Truth for Typescript Types. In: *All Things Typescript* [online]. 2023. Available also from: `https://www.allthingstypescript.dev/p/using-zod-schemas-as-source-of-truth`. Accessed 2024-04-11.

57. JOHNSON, Garrett. *three-mesh-bvh* [online]. 2024. Available also from: `https://github.com/gkjohnson/three-mesh-bvh`. Accessed 2024-04-15.

58. WALTON, Philip. First Contentful Paint (FCP). In: *web.dev* [online]. 2023. Available also from: `https://web.dev/articles/fcp`. Accessed 2024-04-19.

59. NOGUEIRA, Thiago. *Modular Kitchen Sink - Game Ready Asset* [online, 3D model]. Sketchfab, 2018. Available also from: `https://skfb.ly/6xQUA`. Purchased under a standard license; Accessed 2024-05-02.

60. CLOUDFLARE, INC. *Cloudflare Pages* [online]. ©2024. Available also from: `https://pages.cloudflare.com`. Accessed 2024-04-23.

61. NETLIFY. *Scale & Ship Faster with a Composable Web Architecture* [online]. ©2024. Available also from: `https://netlify.com`. Accessed 2024-04-23.

62. CLOUDFLARE, INC. *Cloudflare Workers* [online]. ©2024. Available also from: `https://workers.cloudflare.com`. Accessed 2024-04-23.

63. AMAZON WEB SERVICES, INC. *Serverless functions, FaaS Serverless* [online]. ©2024. Available also from: `https://aws.amazon.com/lambda/`. Accessed 2024-04-23.

64. GOOGLE. *Cloud Functions* [online]. [N.d.]. Available also from: `https://cloud.google.com/functions`. Accessed 2024-04-23.

65. HOMÈS, Bernard. *Fundamentals of Software Testing*. ISTE/Wiley, 2012. ISBN 978-1848213241.

66. KRYSIK, Arkadiusz. *21 Types of Software Testing Every Engineer Should Be Using for Better Results* [online]. Stratoflow, 2023. Available also from: `https://stratoflow.com/types-of-software-testing/`. Accessed 2024-04-27.

67. KHORIKOV, Vladimir. *Unit Testing Principles, Practices, and Patterns*. Manning Publications, 2020. ISBN 978-1617296277.

68. LAMBDATEST INC. *Real Time Browser Testing* [online]. 2024. Available also from: `https://www.lambdatest.com/support/docs/real-time-browser-testing/`. Accessed 2024-04-28.

69. BARNUM, Carol M. *Usability Testing Essentials*. 2nd ed. Morgan Kaufmann Publishers, 2020. ISBN 978-0128169421.

70. MORAN, Kate. *Usability Testing 101* [online]. Nielsen Norman Group, 2019. Available also from: `https://www.nngroup.com/articles/usability-testing-101/`. Accessed 2024-04-28.

71. NIELSEN, Jakob. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993. ISBN 978-0-12-518406-9. Available from DOI: `10.1016/C2009-0-21512-1`.

72. NIELSEN, Jakob. *Why You Only Need to Test with 5 Users* [online]. Nielsen Norman Group, 2000. Available also from: `https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`. Accessed 2024-04-29.

73. BROOKE, John. SUS: A 'Quick and Dirty' Usability Scale. In: *Usability Evaluation in Industry* [ebook]. Taylor & Francis, 1996, chap. 21, pp. 189–194. Available from DOI: `10.1201/9781498710411-35`.

74. SAURO, Jeff. Measuring Usability with the System Usability Scale (SUS). In: *MeasuringU* [online]. 2011. Available also from: `https://measuringu.com/sus/`. Accessed 2024-04-30.

75. APPLE INC. *Magic Mouse - White Multi-Touch Surface* [online]. ©2024. Available also from: `https://www.apple.com/shop/product/MK2E3AM/A/magic-mouse-white-multi-touch-surface`. Accessed 2024-05-05.

Contents of the Attachment

```
readme.txt ................... overview of the content of the attachment
├── builds ......................................... builds of the solutions
│   ├── main/dist .................. build of the configurator web application
│   └── admin/dist .............. build of the administrator web application
├── text ............................................... text of the thesis
│   ├── src ........................ source code of the text in LaTeX format
│   └── thesis.pdf ............................ compiled text of the thesis
└── impl ................. repository with the implementation of the solution
```