

# **3DBUY: A 3D PRODUCT VISUALIZER AND CUSTOMIZER FOR COMMERCE**

**A MAJOR PROJECT REPORT**

*Submitted by*

**WIESLAW SAMUSHONGA**

*in partial fulfillment for the award of the degree  
of*

**BACHELOR OF TECHNOLOGY  
IN  
COMPUTER SCIENCE AND ENGINEERING**



**ALAKH PRAKASH GOYAL SHIMLA UNIVERSITY  
SHIMLA 171009**

**18001694**

**JUNE 2022**

**ALAKH PRAKASH GOYAL SHIMLA  
UNIVERSITY  
SHIMLA 171009**

**BONAFIDE CERTIFICATE**

Certified that this project report **“3DBUY: A 3D PRODUCT  
VISUALIZER AND CUSTOMIZER FOR COMMERCE”** is the  
bonafide work of **“WIESLAW SAMUSHONGA”** who carried out the  
project work under my supervisor.

**SIGNATURE**

**PRASHANSA TANEJA  
SUPERVISOR**

Associate Professor

Department of CSE

**SIGNATURE**

**PRASHANSA TANEJA**

**CLASS MENTOR**

**SIGNATURE**

**PRASHANSA TANEJA**

**HEAD OF THE DEPARTMENT**

## ACKNOWLEDGEMENT

First and foremost, I would like to acknowledge the Almighty God for the successful completion of the project title **3DBuy: A 3D Product Visualizer and Customizer for Commerce**. I would like to say thanks to HBSB Infotech for the opportunity to learn. Special thanks go to my academic supervisor Ms. Prashansa Taneja of CSE Department at APG Shimla University and Er. Himanshu Bansal (CEO and Founder of HBSB Infotech). Many thanks to my technical supervisors Mr. Jatin Sharma (3D Modeller/Animator), Mr. Ankit Sharma (MERN Developer), Mr. Aman Kumar (Systems Admin/DevOps Er) and Mr. Nenyasha Madyavanhu (Web/Python and Android Developer) for their personal efforts, practical skills, professional guidance, and direction towards a successful internship and project delivery. Finally, I would like to extend my heartfelt gratitude to my family members especially my parents, classmates, and friends for their invaluable support throughout my internship.

**Wieslaw Samushonga**  
**(18001694)**

## ABSTRACT

Competition in the commercial industry have reached a point that ordinary businesses cannot make huge turnovers using traditional shops and e-commerce. Dramatic breakthroughs in technology and processing power along with an ever-increasing number of features for devices, have opened doors to a wide range of possible “**wow factors**” in business. One such breakthrough is the arrival of 3D web technologies. Today’s customers **have higher visual expectations than ever.**

**3DBuy** is a powerful *3D product visualizer and configurator* that aims to make buyers happy by availing beautiful product experiences. For businesses, it aims to trigger and create engaging experiences that gets results. **3DBuy** delivers an unrivalled product experience that looks great and scales with any business. Built using the latest cutting-edge 3D and web technologies, it’s for-sure a “business hack”.

## TABLE OF CONTENTS

CHAPTER NO.	CHAPTER NAME	PAGE NO.
	<b>Title Page</b>	<b>i</b>
	<b>Bonafide Certificate</b>	<b>ii</b>
	<b>Acknowledgement</b>	<b>iii</b>
	<b>Abstract</b>	<b>iv</b>
	<b>Table of Contents</b>	<b>v</b>
	<b>List of Figures</b>	<b>viii</b>
1.	<b>CHAPTER 1</b>	<b>1</b>
	<b>Introduction</b>	<b>1</b>
1.1.	Understanding 3DBuy	1
1.2.	Objectives	1
2.	<b>CHAPTER 2</b>	<b>3</b>
	<b>Feasibility Study</b>	<b>3</b>
2.1.	Sell More, Faster, With Less Effort!	3
2.2.	What is Product Configuration?	4
2.3.	Is Product Configuration Relevant for all Market Approaches?	5
2.4.	Different Types of Product Configuration	5
2.4.1.	Geometrical	5
2.4.2.	Functional	6
2.5.	What is Configurability?	7
2.5.1.	Design for Configuration	7
2.5.2.	Configurable Supply Chain	8
2.5.3.	Product Configuration and IT Systems	9
3.	<b>CHAPTER 3</b>	<b>11</b>
	<b>Modularization</b>	<b>11</b>
3.1.	What is Modularization?	11
3.2.	The History of Modularization and Modular Systems	12
3.3.	Software Modularization: Improve Quality, Time-To-Market and Enable Scaling	13
3.4.	Three Great Examples of Modularization and Modular Products	14
3.5.	Four Fundamentals for Modularization Success	17
4.	<b>CHAPTER 4</b>	<b>20</b>
	<b>Tools and Technologies used</b>	<b>20</b>
4.1.	Requirements	20
4.2.	ReactJS	21
4.3.	ThreeJS	21
4.4.	React-Three-Fiber	22
4.5.	React-Three-Drei	22
4.6.	React-Oauth/Google	22
4.7.	React-Facebook-Login	22
4.8.	React-ga	23
4.9.	Dotenv	23

4.10.	File-saver	23
4.11.	Js-cookie	23
4.12.	Jwt-decode	23
4.13.	Lodash	24
4.14.	Three-GLTF-Exporter	24
4.15.	React-Toastify	24
4.16.	Redux	24
4.17.	React-Redux	25
4.18.	Reduxjs/toolkit	25
4.19.	Classnames	25
4.20.	Axios	26
4.21.	Buffer	26
4.22.	JSONWebToken	26
4.23.	SASS	26
4.24.	Material UI Core	27
4.25.	Express	27
4.26.	MongoDB	27
4.27.	Mongoose	28
4.28.	Morgan	28
4.29.	Google-Auth-Library	28
4.30.	Express-Validator	28
4.31.	Nodemailer	29
4.32.	Express-JWT	29
4.33.	Cors	29
4.34.	Body-Parser	29
5.	<b>CHAPTER 5</b>	30
	<b>Design</b>	30
5.1.	Architectural && Technology Design	30
5.2.	3D Product Configuration app (product)	31
5.2.1.	Components	31
5.2.2.	Data Flow Diagram/Flowcharts	33
5.2.3.	State Management Explained.	34
5.2.4.	Interpreting the Product Configurator Workflow Design:	36
5.3.	Landing Page	38
5.4.	Authentication and Authorization app	45
5.4.1.	Auth App Front End Folder Structure	45
5.4.2.	Auth App Server/Back End Folder Structure	47
5.4.3.	Sign up	48
5.4.4.	Account Activation Email with Link	48
5.4.5.	Activate Account Page	48
5.4.6.	Sign In	49
5.4.7.	Forgot Password Page	49
5.4.8.	Email with Password Reset Link	50
5.4.9.	Password Reset Page	51

5.4.10.	Private Profile Page	51
5.5.	Code Snippets	52
5.5.1.	SignUp Form Submit	52
5.5.2.	Account Activation Page: Activate	53
5.5.3.	Sign in Page: Google Login	54
5.5.4.	Forgot Password Page: Submit	55
5.5.5	Password Reset Page: Submit	56
5.5.6.	Profile Update Page: Load Profile	57
6.	<b>CHAPTER 6</b>	58
	<b>Conclusion</b>	58
6.1.	Future Scope	58
	<b>REFERENCES</b>	59

## LIST OF FIGURES

Figure 1: A Modular System .....	11
Figure 2: Standardization and Customization .....	12
Figure 3: Spotify Web API.....	16
Figure 4 : Front End packages .....	20
Figure 5 : Back End Packages .....	21
Figure 6: 3 Tier Architecture .....	30
Figure 7: Base 3D Configurator Components Structure .....	31
Figure 8: File Structure and Names of 3D Configurator .....	32
Figure 9: UI components naming and placement.....	32
Figure 10: Example 1 of Custom Product .....	33
Figure 11: Example 2 of Custom Product .....	33
Figure 12: State Management Overview .....	34
Figure 13: Color and Texture States in Code .....	35
Figure 14: Scene Graph States in Code .....	35
Figure 15: Active Option State in Code .....	36
Figure 16: Download button trigger State in Code .....	36
Figure 17: Download Button onClick State Implementation .....	36
Figure 18: 3D Product Configurator Workflow/DFD .....	37
Figure 19: Landing Page component Structure .....	38
Figure 20: Landing Page Components File Structure .....	38
Figure 21: Landing Page Assets Management File Structuring.....	39
Figure 22: Landing Page Layouts, Sections and Elements Structuring .....	40
Figure 23: Landing Page Header and Hero Section 1 .....	41
Figure 24: Landing Page Hero Section 2 .....	41
Figure 25: Landing Page Features Tile Section .....	42
Figure 26: Landing Page Features Split Section 1 .....	42
Figure 27: Landing Page Features Split Section 2 .....	43
Figure 28: Landing Page Features Split Section 3 .....	43



Figure 29: Landing Page Testimonials Section.....	44
Figure 30: Landing Page Footer and Cta Section.....	44
Figure 31: Auth App React Components Structure: Highlighted .....	45
Figure 32: Auth App React Components File Structuring: Expanded Folders ..	46
Figure 33: Auth App ExpressJS Base Folder Structuring .....	47
Figure 34: Auth App Express JS Server File Structuring.....	47
Figure 35: Sign Up Form with email sent notification.....	48
Figure 36: Email with Account Activation Link .....	48
Figure 37: Account Activation Page .....	49
Figure 38: Sign in Page .....	49
Figure 39: Forget Password Page .....	50
Figure 40: Password Reset Link Email .....	50
Figure 41: Password Reset Page.....	51
Figure 42: Profile Update Page .....	51
Figure 43: Sign Up Form Code Snippet .....	52
Figure 44: Account Activation Page Code Snippet.....	53
Figure 45: Sign in Page Code Snippet.....	54
Figure 46: Forget Password Page Code Snippet .....	55
Figure 47: Password Reset Page Code Snippet.....	56
Figure 48: Profile Update Page Code Snippet.....	57

# CHAPTER 1

## Introduction

### 1.1. Understanding 3DBuy

A picture may start the story, but an interactive 3D configurator lets customers determine their product's shape, color, materials, and accessories. This is the running theme of 3DBuy. It is a product built to be integrated into eCommerce platforms, ERPs, PIMs, and Agencies to improve experiences. It has been in development for 4 months at the time of this report.

The full specification of features is as follows:

#### **Advanced configuration**

- Dynamic product assembly
- Parametric configuration
- Nested configuration

#### **Integrations**

- eCommerce platforms
- ERPs
- PIMs

#### **Agencies**

- Manufacturing outputs
- Bill of materials
- Assembly instructions
- Configure price quotes

#### **3D assets management**

- Product catalogue
- Product, parts, and style storage
- Materials library
- Tasks and approval systems

### 1.2. Objectives

- Sell more by letting customers configure complex products, in real time

- Show every potential product customization, in real time
- Make 3D asset management easier
- Configure highly complex products
- Let customers create products unique to them
- Enter the Metaverse with product NFTs
- Integration into any website with a built-in user interface
- Create a satisfactory “win win” scenario for the buyer and seller

## CHAPTER 2

### Feasibility Study

#### 2.1. Sell More, Faster, With Less Effort!

Henry Ford once said, *"Any customer can have a car painted any color that he wants as long as it is black."* What was once a fundamental enabler for industrialization can feel far away today, where mass customization, modularization, and online product configurators are taken for granted by businesses and consumers alike.

Today customers want to configure their own individual solutions. They don't want a standardized package but a solution that meets their specific needs and preferences. The challenge is to enable this in a process that puts the customer in the front seat but doesn't overwhelm the business with complexity.

Imagine walking into a car dealership. The salesperson greets you nicely and asks some questions about you and what specific needs you have. A good salesperson will tailor the following questions based on each response and utilize all the available information. By asking relevant questions, you feel that the salesperson understands your needs and cares about finding the perfect solution. After a brief conversation, the salesperson presents a car that fulfils all your wishes, including the budget constraints. Of course, the suggested car is one that the dealership can supply and deliver within your required timeframe.



*The salesperson is performing the main function of product configuration: understanding the customer's needs and proposing an attractive solution that fits those needs.*

The imaginative example is a basic description of a salesperson's primary task, initializing the sales to delivery process: meeting potential customers, understanding their needs and requirements, and what is needed to win the order. Translating this into an offer that fulfils the customer's expectation and fits with the business objectives and capabilities of the company is a core competence of any business.

What the salesperson is doing is also an example of product configuration. In this section, I will explain further what Product Configuration is, then move on to analyse for what types of businesses product configuration is relevant, looking into different types of configurators, and how to enable configuration within design engineering, supply chain, and IT systems. I will finish by exploring different types of software for product configuration and looking at some of the main benefits of product configuration.

## **2.2. What is Product Configuration?**

Product Configuration is the function of selecting and arranging parts in a combination that can be produced and delivered to fulfill a specific customer request. Product Configuration can be performed manually by a salesperson or a sales engineer or automatically by a software solution.

Product Configuration is different from tailoring in how the configuration is prepared – the questions asked, and the selection and arrangement of parts follow a predefined logic allowing other business functions to be well-prepared for efficient execution of the order.

Product Configuration is different from a sales catalogue in that the selection is interactive. The salesperson or configurator asks questions and either ask further questions or suggests an attractive solution based on the replies. No complete solution or combination is defined in advance. The offered solution and combination are a result of each specific combination of needs and responses.

## 2.3. Is Product Configuration Relevant for all Market Approaches?

Depending on the how you market your assortment, product configuration will be used differently. Some companies will only use it as an internal tool to generate BOMs, and some will use it to support salespersons. Some will even use it at the front end, replacing the salesperson and making the configuration process self-service.

We like to separate between three different market approaches: Product-based, Configuration-based, and Project-based. For each market approach it is critical to align the back-end ways of working and tools to benefit from product configuration.



The front end is where the company markets, sells, and delivers its products. The back end is the company's internal workings and inbound supply chain.

## 2.4. Different Types of Product Configuration

Depending on the nature of the sold product, the configuration problem could be either Geometrical (how to position items), or Functional (what items to use), or a mix of both.

### 2.4.1. Geometrical

Geometrical configuration is about positioning objects to create a layout of e.g. a production line. The layout must fulfil given rules of functionality (e.g. have the right production sequence). The layout must also be free from collisions, both internally in the configured product, but also with any existing objects such as pillars, other

equipment, etc. A typical task of a geometrical configurator is also to adjust parametric objects, perhaps to bridge the gap between two machines with a conveyor.

CAD software with predefined functional blocks or a geometrical configurator could be the tool to use.

Two examples of publicly available geometrical configurators are IKEA's Kitchen Planner, and Elfa's Storage Planner.

Geometrical configurators typically give instant visual feedback on the validity of the layout.



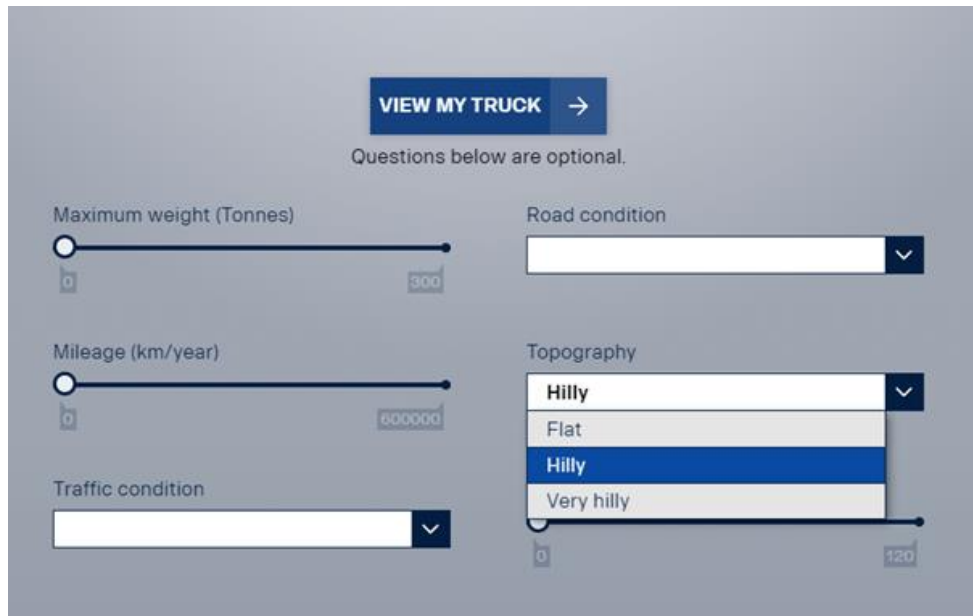
*IKEA's Kitchen Planner is an excellent example of a geometrical configurator. Like most geometrical configurators, it also performs functional configuration of the positioned items.*

### **2.4.2. Functional**

Functional configuration is about selecting a correct solution given a set of requirements or needs. Based on the required performance, functions, and features, the configurator selects and dimensions the different parts of the products (modules) and ensures that they are compatible and that the combination of parts performs according to the requirements. The input to the configurator can be needs-based, e.g., enter your electricity cost and get the optimal configuration. The input can also be solution-based,

where the input is a selection of available variants or options, and the user must decide the best choice.

An example of a publicly available functional product configurator is Scania's truck configurator.



The image shows a screenshot of the Scania Truck Configurator interface. At the top, there is a blue button labeled "VIEW MY TRUCK" with a right-pointing arrow. Below this button, the text "Questions below are optional." is displayed. The interface features several input fields and sliders:

- Maximum weight (Tonnes):** A horizontal slider with a circular handle, ranging from 0 to 300.
- Mileage (km/year):** A horizontal slider with a circular handle, ranging from 0 to 600,000.
- Traffic condition:** A dropdown menu with a downward arrow.
- Road condition:** A dropdown menu with a downward arrow.
- Topography:** A dropdown menu with a downward arrow, showing a list of options: "Hilly" (selected), "Flat", "Hilly", and "Very hilly".

*Scania's Truck Configurator is a nice example of a functional configurator.*

## 2.5. What is Configurability?

Configurability is a measurement of how suitable the product is for configuration. It is primarily linked to:

1. How the product is designed and structured.
2. How the supply chain is set up to support a configurable product.
3. How the product is represented in IT systems

When understood together, configurability and configuration enable the mass customization of products to meet specific and individual customer needs efficiently and effectively.

### 2.5.1. Design for Configuration

To create configurable designs, products should be flexible enough to allow for the adding, removing, or replacing of elements without impact across the product. Changes



must be isolated to the directly customized element, without causing indirect changes to surrounding elements.



*A modular design*

A modular design has exactly this ability. Why? Because functions, features, and performance levels are encapsulated in individual modules and the modules themselves are protected from each other by interfaces. This allows one module, or variants of one module, to be changed while still fitting the interface, without changing any other module. The more modular the product is, the easier to configure.

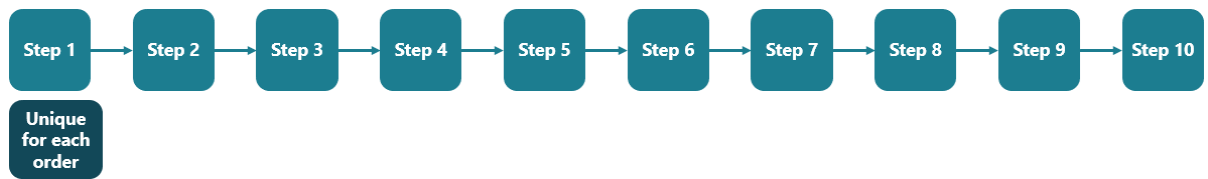
A modular product will always be configurable. However, the opposite is not necessarily true, a configurable product is not necessarily modular. For this case, setting up the configuration model will be much more complex and harder to maintain since the different products will likely have different logics. And components will be product-specific rather than reusable across different products.

### **2.5.2. Configurable Supply Chain**

Setting up the product design for configuration is not enough if the supply chain is not prepared to handle the flexibility.

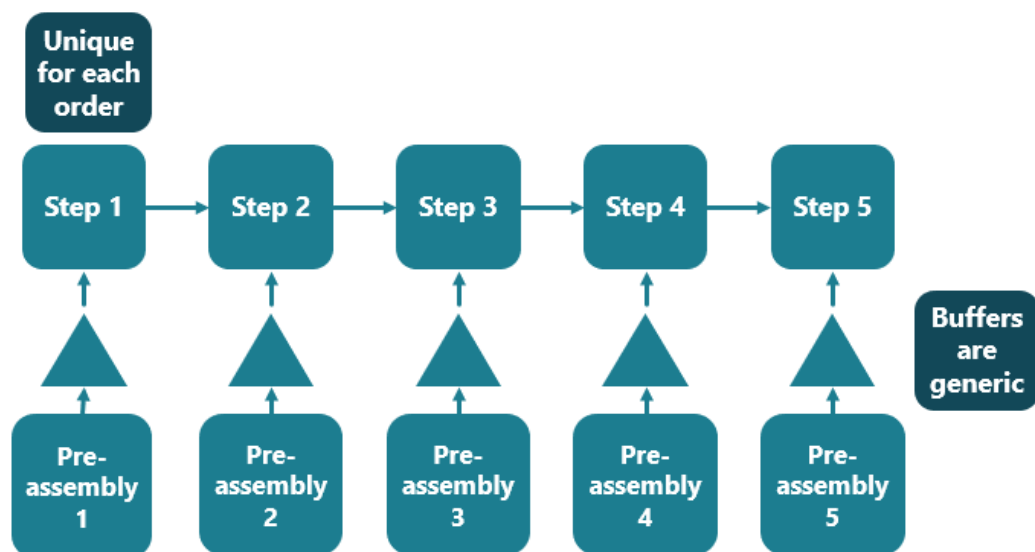
Many traditional industrial supply chains are set up to make products to stock. Batching, low-cost country sourcing, and dedicated lines are concepts that make standard products more efficient to produce – but these concepts are not flexible to handle mass customization.

Imagine a supply chain with 10 process steps. If the unique customer request affects already the first step, nothing can be prepared in the advance. The product cannot be delivered until all 10 process steps have been carried out.



In a process where the first step is unique for each order, the whole process must be carried out before the product can be delivered.

To solve this, we need to delay the point of variance. This is the point in the supply chain where parts and assemblies become a unique order combination, instead of generic parts that could fit into multiple combinations. By delaying the variance point, all assembly operations that make the combination order unique are delayed until the order is received. At this point, the requested combination is known and assembled to order. Only generic parts up to module level are purchased or produced to stock, before the order, and are then ready to be assembled in the correct combination.

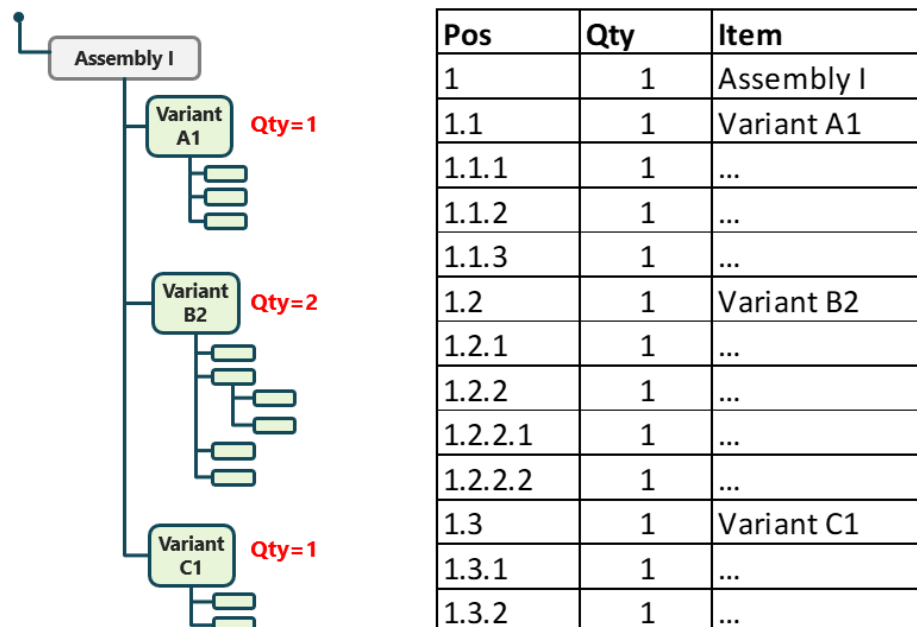


By pre-assembling generic parts that could fit in multiple configurations, the order-to-delivery lead time can be drastically reduced. The point of variance has been moved later, and there are only five steps to carry out before being able to deliver the product.

### 2.5.3. Product Configuration and IT Systems

A customizable product needs to be represented in IT systems so that elements can be easily added, removed or replaced.

One important aspect of configurability is the level at which parts are documented in the bill of materials (BOMs). Many companies sub-optimize the BOM to simplify or reduce part number count. Parts are then documented on a too high level, and above the level at which customers want to add, remove, or change elements. This means that only predefined combinations of elements exist, and no unplanned combinations can be made. Over time, the goal of part number count reduction becomes harder since the number of combinations needed grows exponentially with a multiplication effect. At this stage, companies are still only making predefined combinations and configurability suffers.

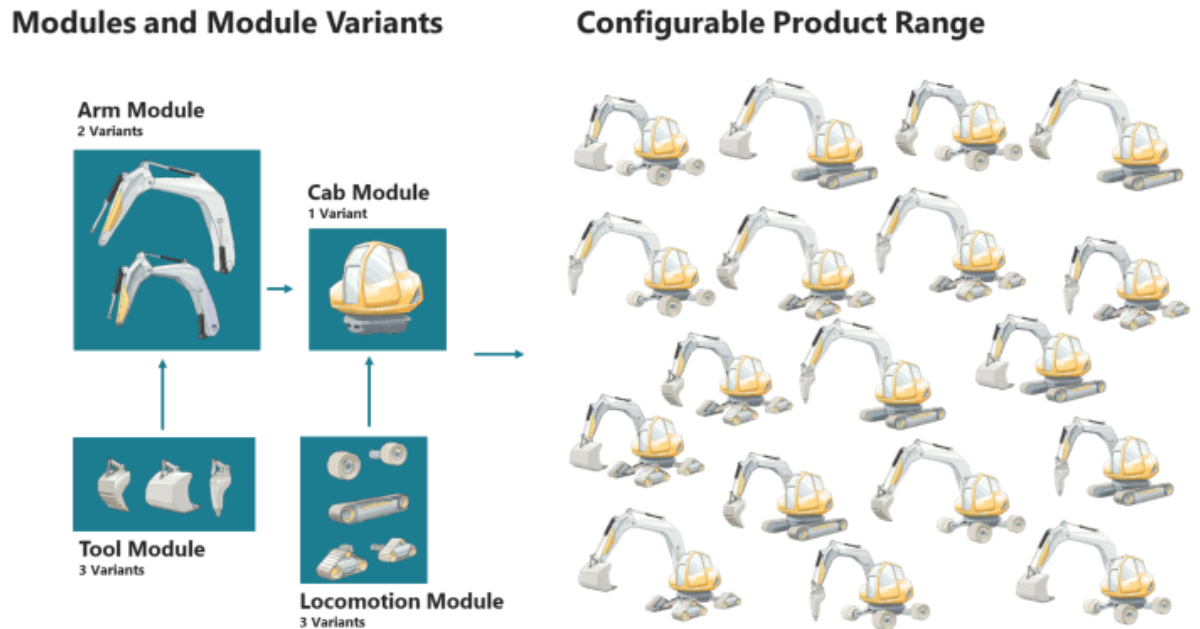


*Two representations of the same multi-level BOM: Graphical and tabulated*

Another important aspect is how to manage the total configurable product range in terms of the bill of materials. Do you have multiple super BOMs for different products? If so, this means you can make changes inside one super-BOM, e.g. add, remove or change elements, but for other changes, you have to change to another super-BOM and throw away what you just did, open up a new super BOM and start from scratch. A truly modular BOM, on the other hand, can handle the full range of products. In the following section, I will further explain the importance of efficient product structure.

## CHAPTER 3

### Modularization



*Figure 1: A Modular System*

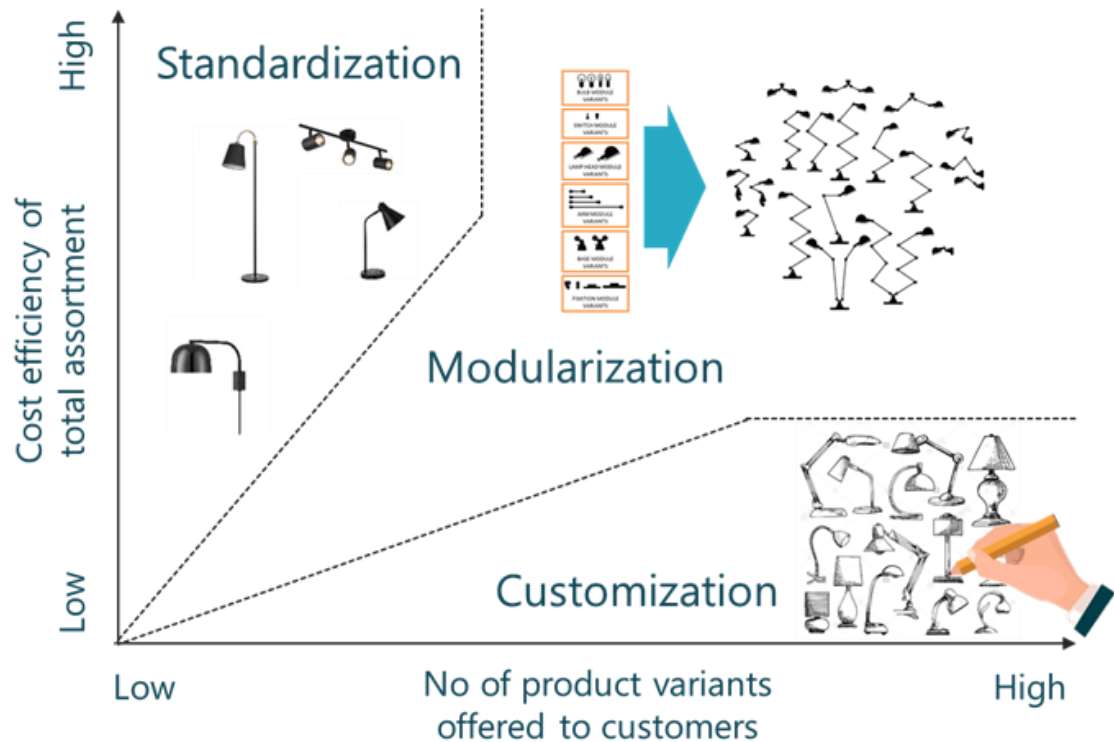
*A Modular System enables the configuration of many products with a limited number of module variants.*

#### 3.1. What is Modularization?

Modularization is the activity of dividing a product or system into modules that are interchangeable. The target of modularization is to create a system that is flexible to create different requested configurations, while reducing the number of unique building blocks (module variants) needed to do so. By re-using module variants across multiple configurations, volumes are consolidated on module level, and economy of scale is reached without standardizing the product.

This way a company can increase the perceived customer value (configuring the right product), increase the speed of development (innovating in one module in a system rather than a whole product), while reducing the internal complexity (limiting the number of variants to design, manufacture, and maintain).

Modularization is combining the benefits of standardization (low cost of complexity) with customization (be able to create the right product for each customer)



*Figure 2: Standardization and Customization*

Modularization captures the benefits of Standardization and Customization

While the most obvious examples of modularization might be physical products, the technique is just as useful within software and service products.

The main enabler of modularization is standardized interfaces. The interface is what enables one variant of a module to be interchanged with another to adapt or develop the performance level or function of the product.

### **3.2. The History of Modularization and Modular Systems**

While proof of Modularization can be traced as far back as the Terracotta Army (200 BCE), modern industrial use of modularization was pioneered by leaders in different industries such as Scania, Toyota, Nippondenso, Dell, and Sony. These companies managed to take clear market leadership using modularization, governing, and improving their Modular Systems over time.

In Germany, modular systems are often called Baukasten, a term literally translated to construction set. Volkswagen is, for example, calling their modular systems Baukasten, e.g., Volkswagen MQB, which stands for *Modularer QuerBaukasten*, roughly translated to Transversal Engine Modular System. Baukasten was frequently used as a term already in the 1930s for do-it-yourself kits, such as model railways, modular radios, and electronics toolkits for kids. These toys have much in common with industrial modular product platforms: They have the same target to create flexibility and the same means - standardizing interfaces.

Sony utilized modularization in a structured way for both the Handycam and Walkman product lines to continuously stay ahead of competition throughout the 1980s. As soon as competitors caught up, Sony could immediately launch the next version and again be the product leader.

Scania, on the other hand, understood that modularity was the way to enable mass customization. By creating a modular system for the truck on different levels, an almost infinite number of variants could be configured – but still assembled on the same assembly lines without costly changeovers. For decades Scania's unique focus on modularity enabled them to outperform the competition regarding profitability by far, even though the volumes were lower than some competitors' volumes. By embracing variance and customization and optimizing for it, Scania successfully reached an excellent economy of scale.

In today's modern economy, successful companies are champions of modularity, both regarding hardware and software, products, and services. With the birth of Web 2.0, modularity reaches across company barriers; micro-services, web apps, and open APIs enable a whole new set of customization and agility possibilities. And as more and more functionality transitions from hardware to software, traditional hardware companies can also benefit from adopting new ways of thinking and working.

### **3.3. Software Modularization: Improve Quality, Time-To-Market and Enable Scaling**

Many companies suffer from complex software development:

- It is hard to scale the development capacity.
- Software is driving quality problems even though the testing efforts are very high.

- It is hard to plan and budget software development since changes lead to unforeseen ripple effects.

Even companies that traditionally have been focused on mechanics are becoming software companies whether they like it or not. CTO's of "hardware companies" realize that they have more software engineers than hardware engineers. Three typical root causes for software architecture problems are "Developed by me" software, highly coupled software structures, and multiple overlapping software platforms.

Making software modules independent, reusable, and interchangeable, software development can be much more efficient and truly agile. Independent modules make development teams autonomous for improved efficiency and time to market.

### **3.4. Three Great Examples of Modularization and Modular Products**

#### **Example 1: Husqvarna Electric Trimmers**

Husqvarna are real performers. Not only in the forest but also in the field of modularization. One example is the modular system for electric trimmers. Multiple brands and an evolving array of SKUs share the same architecture, supporting the transformation from corded trimmers to battery powered.



*Husqvarna Electric Trimmers*

This illustration shows how a modular system can be flexible enough to differentiate brand characteristics without losing scale benefits in functional parts. Components such

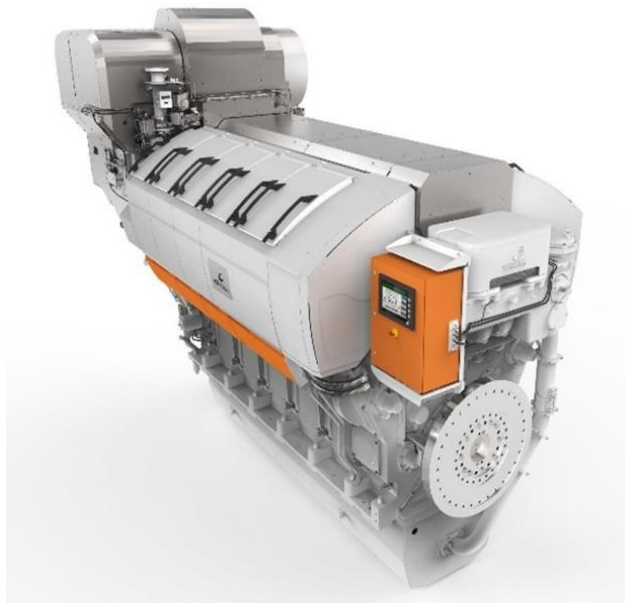
as motors, batteries, speed control, and cabling can have substantial scale advantages. And when the commonality can even be made on the sub-assembly level, it opens up possibilities for sourcing larger, pre-tested assemblies.

Other great examples from Husqvarna can be seen in robotic mowers where multiple brands share architectures, while configuration rules and styling modules separate brands and price points.

Husqvarna is also pioneering in using IoT and cloud solutions to push the business model from selling hardware and consumables to selling gardening tools as a service with their *Tools for You* concept. IoT is also used for professional fleet management.

### **Example 2: Wärtsilä 4-Stroke Engines**

Wärtsilä makes enormous engines, and they have a vast portfolio of them. They produce from what are relatively small engines (around 200 mm cylinder bore) used in for example auxiliary or diesel-electric power generators or on ships to huge powerhouses (500 mm+ cylinder bore) used in power plants and direct drive on ships.



Wärtsilä 4-Stroke Engines

Requirements are high regarding flexibility – machine room layouts, fuel types, power requirements, and operational patterns, just to name a few driving forces. At the same time, technology is pushed, both by competitors but even more so by ever-increasing sustainability requirements both from nations and from the United Nations. Wärtsilä's modular 4-stroke engine system is prepared to take on the challenge. Some great

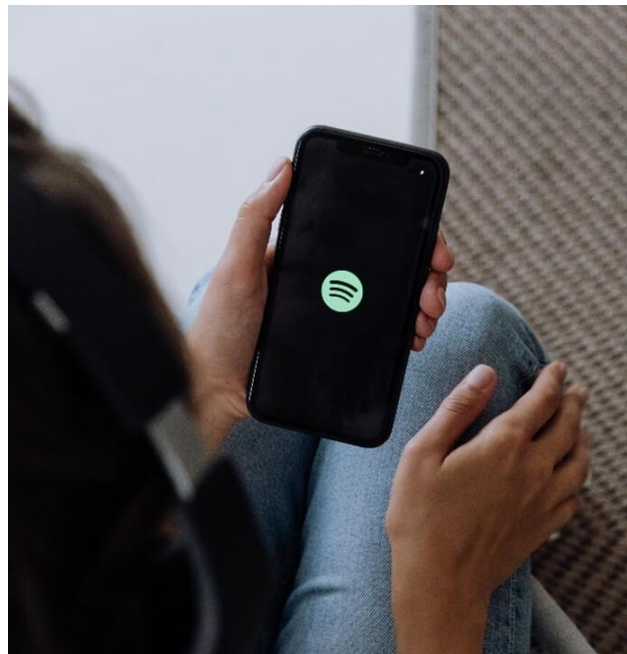


examples of modularization: By making the engine block flexible in the positioning of the main power take out shaft and the turbo charger assembly, machine room flexibility is reached without duplicating complexity. By creating standardized interfaces between fuel injection systems and cylinder heads, fuel types can be configured without affecting the engine's fundamental functional parts.

What's more? Wärtsilä 31 is the most fuel-efficient 4-stroke engine available on the market. Great for the operator, great for the planet. Read our full case story of the [The World's Most Efficient Engine](#).

### **Example 3: Spotify Web API**

By [publishing a public API](#) that any developer can use, Spotify is pushing the barrier and limit of their product.



*Figure 3: Spotify Web API*

Even greater products are created by infusing Spotify modules, such as Albums, Libraries, and Playlists. This open-access means that Spotify can rapidly become an integrated part of Smartwatches, Smart speakers, Car entertainment systems, TVs – and so on. These developments would never have been possible at this rate without cooperation. The key here is standardized interfaces.

By publishing well-specified and stable interfaces, the development can be made concurrently on both sides of the interfaces, completely without communication. This concurrent development is the same effect that modularization brings within a

company. However, the challenge of specifying the interfaces may be even more critical. If care is not taken, businesses may end up maintaining many different generations of their APIs.

What's great about the Spotify example is that it removes the organizational borderline for genuinely global cooperation. Modularization does not have to be an internal strategy only.

### **3.5. Four Fundamentals for Modularization Success**

A chain is no stronger than its weakest link. To reach sustainable success within Modular Systems, you must ensure strong competence in four areas.

#### **Modular System Strategy**

Understanding how your Modular platforms are supporting the business strategy is fundamental for success. Without alignment on present challenges, future visions, and what path you are on, you will only reach a fraction of your true potential. Alignment must start from the top and ripple down in the company.

Alignment must be secured top-down for how to:

- Connect Business Strategy to Product Architecture Strategy
- Organize and focus Innovation
- Understand the Business potential of modularization

Product architecture strategies may be very different across companies and business areas. One example is the construction industry's trend to use modular building systems to move from project-based construction to configuration-based construction. The target is two-fold: Improve customer value and increase efficiency. Each business needs to correctly understand where they are and how far they want to reach to steer efficiently. A tool to analyze this is the Product Architecture Maturity Model

Another example is the fight for sustainability. How can Modular Design help companies embrace the circular economy? Modular Management partner Colin de Kwant is researching the topic in cooperation with the Royal Institute of Technology.

#### **Modularization: Creating a Modular System**

Creating a Modular System requires understanding what requirements the products must fill, how these requirements might change over time, and what performance levels must be reached to satisfy the customers. Market models should be created to ensure that the right focus and prioritization are set. Tools that can help in the process are, for example, needs-based segmentation. Scott Jiran has written a great blog post on Customer Canvas as a tool to analyze customer benefits of different segments.

The best Modular Systems have the right flexibility and agility – to create that requires deep knowledge about how the product and the technology within must be adapted and developed to produce the needed functionality and performance.

But understanding the technology is not enough - understanding how business strategies across all functions should be enabled by the products is also necessary. One example is production strategies of implementing lean. How should we implement lean and modularity to reach winning synergies?

Understanding the needs of the market, technology, and business is a great start for modularization with the right flexibility and agility. But to ensure that re-use and long-term governance can be achieved, the modules and interfaces must also be designed and documented to enable the architecture intention to be kept.

### **Improving a Modular System**

All companies that have products in production have a product architecture, modular or not. This means that all companies can improve their existing architecture.

Incremental improvements, continuous improvement, continual improvement, Kaizen – we have many names for the things we love. Great results can typically be achieved by improving over time. This is also one of the key properties of a great modular system – it lasts for a long time, and therefore can be improved over a long time. Many topics are of interest in this area of improving the product architecture, some examples:

- How to analyze product portfolios for pruning. Prioritize R&D and align phase-in/phase-out plans across the company. One example: Cutting the tail of unprofitable products, taking cost of complexity into account.
- Identifying and prioritizing candidates for redesign, optimization, and value engineering. Set targets, act, follow-up.
- Identifying and prioritizing supply chain actions to reach Procurement Excellence. Consolidate volumes, align module and supply chain strategies. The Covid-19 pandemic has also increased the spotlight on resilient supply

chains even more. How can modularization enable geo-flexibility and dual sourcing?

### **Core Capabilities Needed to Succeed with Modularization**

The fourth and last fundament is the core capabilities of the company. No matter how good your product is, you will fail if you don't have the capabilities in place to utilize and maintain it:

- Processes, roles, decision models to achieve efficient modular system governance, design, sales, sourcing, and production.
- Improve efficiency in Sales to Delivery & Develop and Maintain processes with digital solutions for sales, product management, R&D, and supply. Companies today are already digital, but how must the systems be adapted to go from products to modular architectures? One example of utilizing modular platforms in the sales system is implementing guided selling tools or configurators.
- How to understand the cost of complexity and balance re-use to direct cost in architecture and design decision

Lars Gullander has written a great blog post titled *How to Manage Change in your Modular Product Architecture?* In it, you will also find examples of decision models for governance.

# CHAPTER 4

## Tools And Technologies Used

### 4.1. Requirements

Operating System: Cross-Platform

Up-to-date Browser on any device.

Processor: 800 MHz or above

RAM (SD/DDR): 1GB

HDD/ SSD: 30GB

IDE: **VSCode** OR Sublime Text OR IntelliJ IDEA Community/ Ultimate  
[recommended]

Deployments : Heroku, Glitch

```
"dependencies": {
  "@material-ui/core": "^4.12.4",
  "@react-oauth/google": "^0.2.1",
  "@react-three/drei": "^9.7.1",
  "@react-three/fiber": "^8.0.14",
  "@testing-library/jest-dom": "^5.16.4",
  "@testing-library/react": "^13.1.1",
  "@testing-library/user-event": "^13.5.0",
  "axios": "^0.26.1",
  "buffer": "^6.0.3",
  "classnames": "^2.2.6",
  "dotenv": "^16.0.0",
  "file-saver": "^2.0.5",
  "js-cookie": "^2.2.1",
  "jsonwebtoken": "^8.5.1",
  "jwt-decode": "^3.1.2",
  "lodash": "^4.17.15",
  "react": "^18.1.0",
  "react-dom": "^18.1.0",
  "react-facebook-login": "^4.1.1",
  "react-ga": "^2.7.0",
  "react-redux": "^8.0.1",
  "react-router-dom": "^5.1.2",
  "react-scripts": "4.0.3",
  "react-toastify": "^5.5.0",
  "redux": "^4.2.0",
  "sass": "^1.49.7",
  "three": "^0.140.2",
  "three-gltf-exporter": "0.0.1"
},
```

*Figure 4 : Front End packages*

```
"dependencies": {
  "body-parser": "^1.19.0",
  "cors": "^2.8.5",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "express-jwt": "^5.3.1",
  "express-validator": "^6.4.0",
  "google-auth-library": "^8.0.2",
  "googleapis": "^100.0.0",
  "jsonwebtoken": "^8.5.1",
  "jwt-decode": "^3.1.2",
  "mongoose": "^5.9.6",
  "morgan": "^1.10.0",
  "node-fetch": "^2.6.7",
  "nodemailer": "^6.7.5",
  "nodemon": "^2.0.16"
}
```

Figure 5 : Back End Packages

## 4.2. ReactJS

React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding.

## 4.3. ThreeJS

Three.js is a wrapper that sits on top of WebGL (Web Graphics Library) and obfuscates all the bulk of the code that would be quite difficult to write yourself from scratch. To quote directly from the three.js documentation:

*WebGL is a very low-level system that only draws points, lines, and triangles. To do anything useful with WebGL generally requires quite a bit of code and that is where three.js comes in. It handles stuff like scenes, lights, shadows, materials, textures, 3D math, etc.*

## 4.4. React-Three-Fiber

React-Three-Fiber is a React renderer for three.js on the web and react-native

So, what exactly is a renderer? Renderers manage how a React tree turns into the underlying platform calls. That means we can build 3D artifacts in the form of reusable React components (staying in the React paradigm) and react-three-fiber will do the job of converting those components to the corresponding three.js primitives.

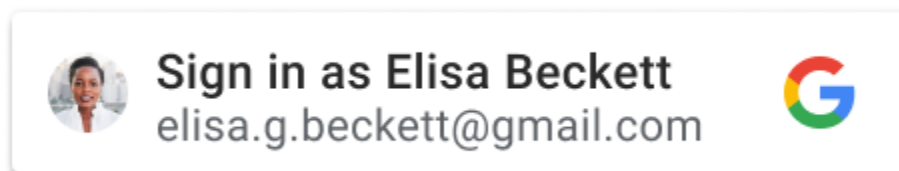
## 4.5. React-Three-Drei

A growing collection of useful helpers and fully functional, ready-made abstractions for @react-three/fiber. If you make a component that is generic enough to be useful to others, think about making it available here through a PR!

## 4.6. React-Oauth/Google

Google OAuth2 using the new Google Identity Services SDK for React. Seamless sign-in and sign-up flows, One-tap sign-up and Automatic sign-in

Add a personalized and customizable sign-up or sign-in button to your website.



*Google Authentication Button*

## 4.7. React-Facebook-Login

A Component React for Facebook Login. You can perform quick sign ups and sign in by authenticating via the Facebook API. You can use the given facebook button , but most importantly create your own and style it accordingly.



*Continue with Facebook Button*

## 4.8. React-ga

This is a JavaScript module that can be used to include Google Analytics tracking code in a website or app that uses **React** for its front-end codebase. It does not currently use any React code internally, but has been written for use with a number of Mozilla Foundation websites that are using React, as a way to standardize our GA Instrumentation across projects.

## 4.9. Dotenv

Dotenv is a zero-dependency module that loads environment variables from a .env file into **process.env**. Storing configuration in the environment separate from code is based on **The Twelve-Factor App** methodology.

## 4.10. File-saver

FileSaver.js is the solution to saving files on the client-side, and is perfect for web apps that generates files on the client, However if the file is coming from the server it is recommended to first try to use **Content-Disposition** attachment response header as it has more cross-browser compatibility.

## 4.11. Js-cookie

A simple, lightweight JavaScript API for handling cookies which works in all browsers, accepts any character, is heavily tested, has no dependency, supports ES modules, supports AMD/CommonJS, is RFC 6265 compliant, has a useful Wiki, enables custom encoding/decoding, is **< 800 bytes** gzipped!

## 4.12. Jwt-decode

**jwt-decode** is a small browser library that helps decoding JWTs token which are Base64Url encoded.

**IMPORTANT:** This library doesn't validate the token, any well formed JWT can be decoded. You should validate the token in your server-side logic by using something like **express-jwt**, **koa-jwt**, **Owin Bearer JWT**, etc.



### 4.13. Lodash

Lodash is a popular javascript based library which provides 200+ functions to facilitate web development. It provides helper functions like map, filter, invoke as well as function binding, javascript templating, deep equality checks, creating indexes and so on.

Lodash can be used directly inside a browser and also with Node.js. Working with objects using JavaScript can be quite challenging, specifically if you have lots of manipulation to be done with them. Underscore comes with lots of features that eases your work with objects.

### 4.14. Three-GLTF-Exporter

An [Three.js](#) exporter for **glTF 2.0**. **glTF** (GL Transmission Format) is an open format specification for efficient delivery and loading of 3D content. Assets may be provided either in JSON (.gltf) or binary (.glb) format. External files store textures (.jpg, .png) and additional binary data (.bin). A glTF asset may deliver one or more scenes, including meshes, materials, textures, skins, skeletons, morph targets, animations, lights, and/or cameras.

### 4.15. React-Toastify

React-Toastify allows you to add notifications to your app with ease. No more nonsense! Easy to set up for real, you can make it work in less than 10sec! Super easy to customize.

RTL support. Swipe to close 🖐️. Can choose swipe direction. Super easy to use an animation of your choice. Works well with animate.css for example : Can display a react component inside the toast!

### 4.16. Redux

Redux is a predictable state container for JavaScript apps. (Not to be confused with a WordPress framework – **Redux Framework**.)

It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great developer experience, such as **live code editing combined with a time traveling debugger**.

You can use Redux together with **React**, or with any other view library. It is tiny (2kB, including dependencies).

## 4.17. React-Redux

React-redux is a state management tool that makes it easier to pass states from one component to another irrespective of their position in the component tree and hence prevents the complexity of the application.

**Working Procedure:** When a react application holds various components in need of state from other components it becomes difficult to realize where the state should reside among these components to make it easier to maintain. React-redux provides a **store** which makes the state inside components easier to maintain. Along with stores, react-redux introduces **actions** and **reducers** which work simultaneously with stores to make the state more predictable.

## 4.18. Reduxjs/toolkit

**Redux Toolkit** is an opinionated, batteries-included toolset for efficient Redux development. It is intended to be the standard way to write Redux logic, and we strongly recommend that you use it.

It includes several utility functions that simplify the most common Redux use cases, including store setup, defining reducers, immutable update logic, and even creating entire "slices" of state at once without writing any action creators or action types by hand. It also includes the most widely used Redux addons, like Redux Thunk for async logic and Redux Select for writing selector functions, so that you can use them right away.

## 4.19. Classnames

A simple JavaScript utility for conditionally joining classNames together.

**Usage with React.js:**

This package is the official replacement for classSet, which was originally shipped in the React.js Addons bundle. One of its primary use cases is to make dynamic and conditional className props simpler to work with (especially more so than conditional string manipulation).

## 4.20. Axios

Axios is a simple promise based HTTP client for the browser and node.js. Axios provides a simple to use library in a small package with a very extensible interface. Lastly, Axios can be used in both the browser and Node.js. This facilitates sharing JavaScript code between the browser and the back end or doing server-side rendering of your front-end apps.

## 4.21. Buffer

With **browserify**, simply require('buffer') or use the Buffer global and you will get this module. The goal is to provide an API that is 100% identical to **node's Buffer API**. Manipulate binary data like a boss, in all browsers! Super fast. Backed by Typed Arrays (Uint8Array/ArrayBuffer, not Object). Extremely small bundle size (**6.75KB minified + gzipped**, 51.9KB with comments). Excellent browser support (Chrome, Firefox, Edge, Safari 11+, iOS 11+, Android, etc.)

## 4.22. JSONWebToken

An implementation of **JSON Web Tokens**. This was developed against draft-ietf-oauth-json-web-token-08. It makes use of **node-jws**.

**jwt.sign(payload, secretOrPrivateKey, [options, callback])**

(Asynchronous) If a callback is supplied, the callback is called with the err or the JWT.

(Synchronous) Returns the JsonWebToken as string

payload could be an object literal, buffer or string representing valid JSON.

## 4.23. SASS

A pure JavaScript implementation of **Sass**. **Sass makes CSS fun again.**

This package is a distribution of **Dart Sass**, compiled to pure JavaScript with no native code or external dependencies.

It provides a command-line sass executable and a Node.js API.

## 4.24. Material UI Core

**Material UI** is the most powerful and efficient tool to build an Application by adding Designs and Animations and use it with technical and scientific innovation. It is basically a design language that was developed by Google in 2014. It uses more Design and Animations, grid-system and provides shadows and lightning effects.

It can be used with all the JavaScript frameworks like **AngularJS**, **VueJS**, and libraries like **ReactJS**, to make the Application more amazing and responsive. With over 35,000 stars on the GitHub, Material UI is one of the top User Interface libraries for React.

## 4.25. Express

ExpressJS is a web application framework that provides you with a simple API to build websites, web apps and back ends. With ExpressJS, you need not worry about low level protocols, processes, etc.

Express provides a minimal interface to build our applications. It provides us the tools that are required to build our app. It is flexible as there are numerous modules available on **npm**, which can be directly plugged into Express.

## 4.26. MongoDB

MongoDB is a schema-less NoSQL document database. It means you can store JSON documents in it, and the structure of these documents can vary as it is not enforced like SQL databases. This is one of the advantages of using NoSQL as it speeds up application development and reduces the complexity of deployments.

## 4.27. Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. Object Mapping between Node and MongoDB managed via Mongoose

## 4.28. Morgan

HTTP request logger middleware for node.js Named after Dexter, a show you should not watch until completion.

### API

```
var morgan = require('morgan')
```

### **morgan(format, options)**

A morgan logger middleware function using the given format and options. The format argument may be a string of a predefined name, a string of a format string, or a function that will produce a log entry.

The format function will be called with three arguments tokens, req, and res, where tokens is an object with all defined tokens, req is the HTTP request and res is the HTTP response. The function is expected to return a string that will be the log line, or undefined / null to skip logging.

## 4.29. Google-Auth-Library

This is Google's officially supported node.js client library for using OAuth 2.0 authorization and authentication with Google APIs.

## 4.30. Express-Validator

Express-validator is a set of express.js middlewares that wraps validator.js validator and sanitizer functions.

### 4.31. Nodemailer

**Nodemailer** is a module for Node.js applications to allow easy as cake email sending. The project got started back in 2010 when there was no sane option to send email messages, today it is the solution most Node.js users turn to by default.

### 4.32. Express-JWT

This module provides Express middleware for validating JWTs (JSON Web Tokens) through the jsonwebtoken module. The decoded JWT payload is available on the request object.

### 4.33. Cors

CORS is a node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options.

**Cross-Origin Resource Sharing (CORS)** is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

### 4.34. Body-Parser

Node.js body parsing middleware.

Parse incoming request bodies in a middleware before your handlers, available under the req.body property.

**Note** As req.body's shape is based on user-controlled input, all properties and values in this object are untrusted and should be validated before trusting. For example, req.body.foo.toString() may fail in multiple ways, for example the foo property may not be there or may not be a string, and toString may not be a function and instead a string or other user input.

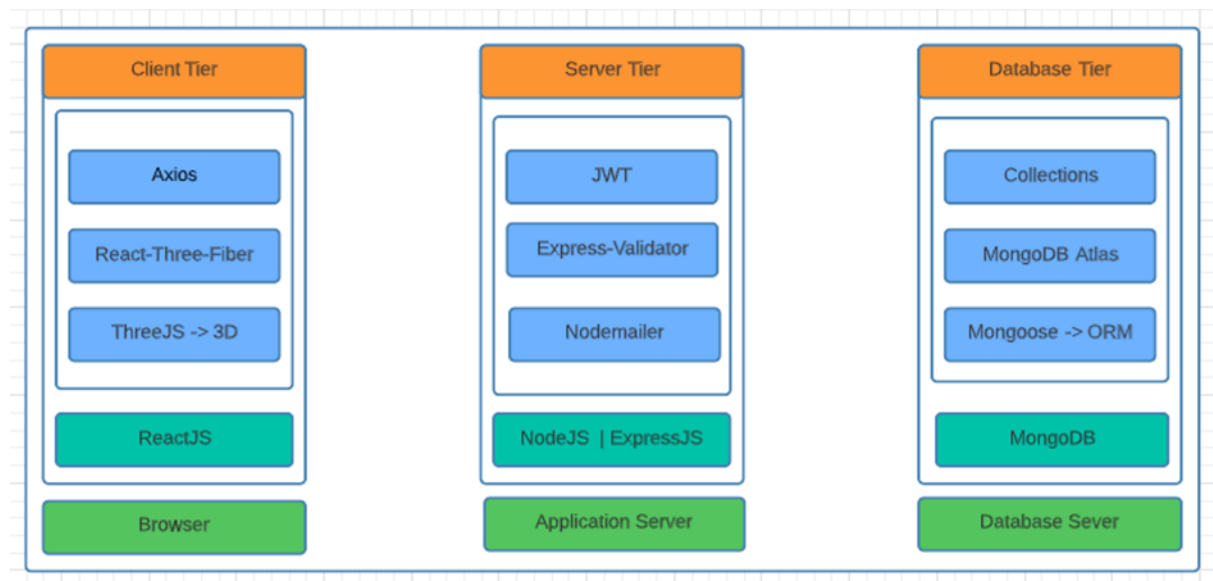
All these technologies were learnt incrementally. 3DBuy is a MERN Stack Product Configurator and Three.js. It only takes understanding each library to appreciate software development. Let's move on to the design of **3DBuy**.

## CHAPTER 5

### Design

#### 5.1. Architectural & Technology Design

A three-tier architecture has been used for this application. It is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms.



*Figure 6: 3 Tier Architecture*

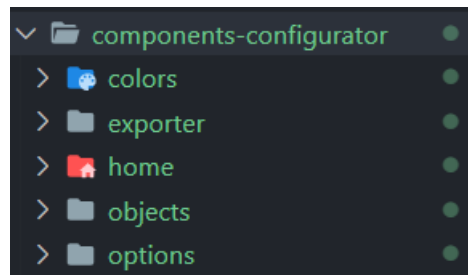
Due to the scalability of this project, it was developed by combining the following component projects:

1. 3D Product Configuration app (**product**)

2. Landing Page
3. Authentication and Authorization app

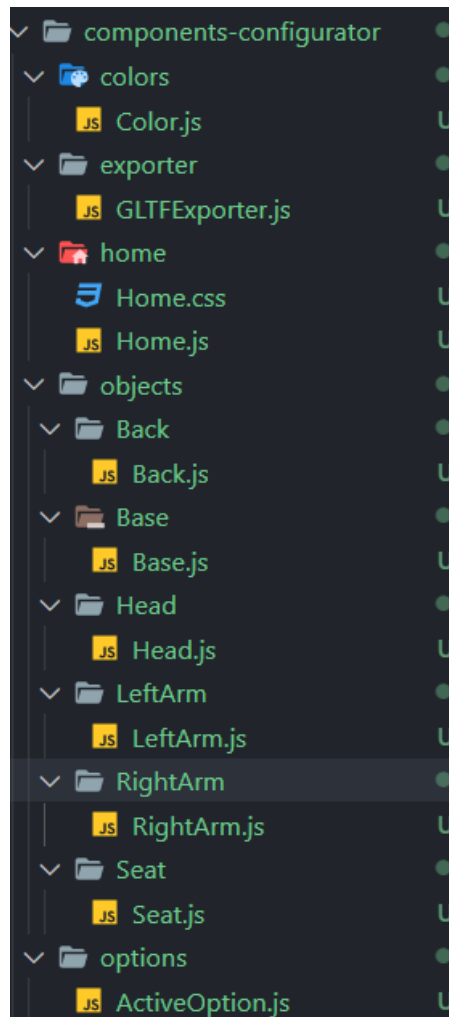
## 5.2. 3D Product Configuration app (product)

### 5.2.1. Components

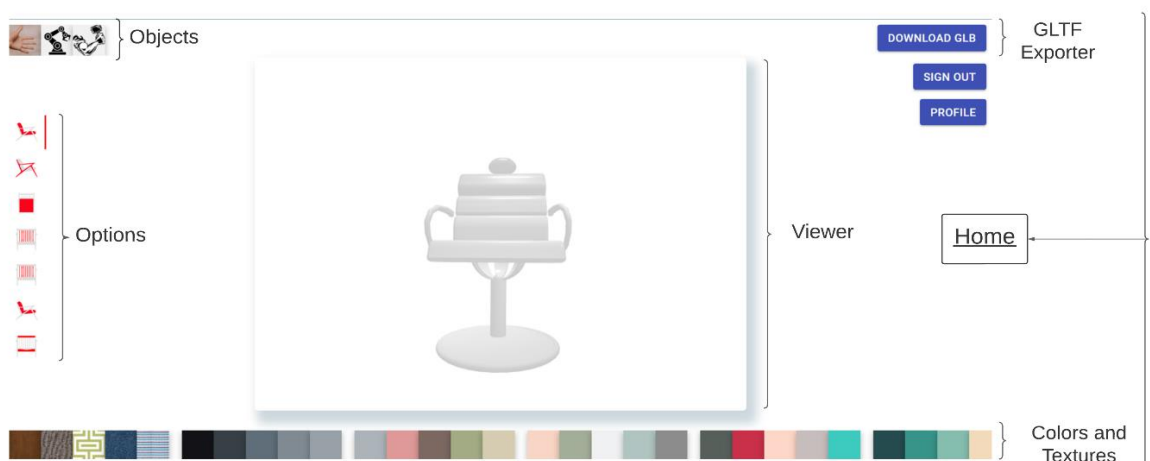


*Figure 7: Base 3D Configurator Components Structure*

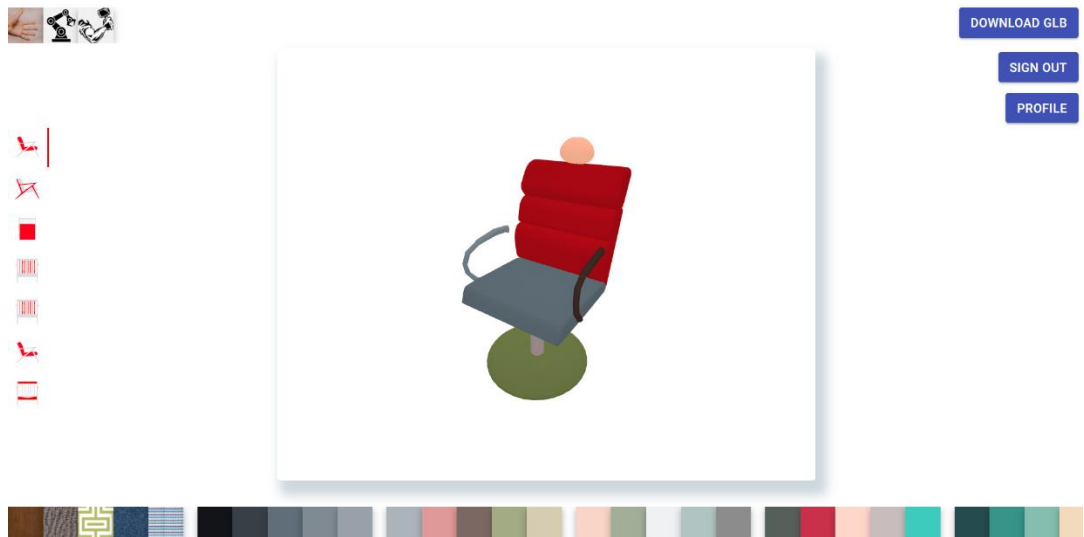




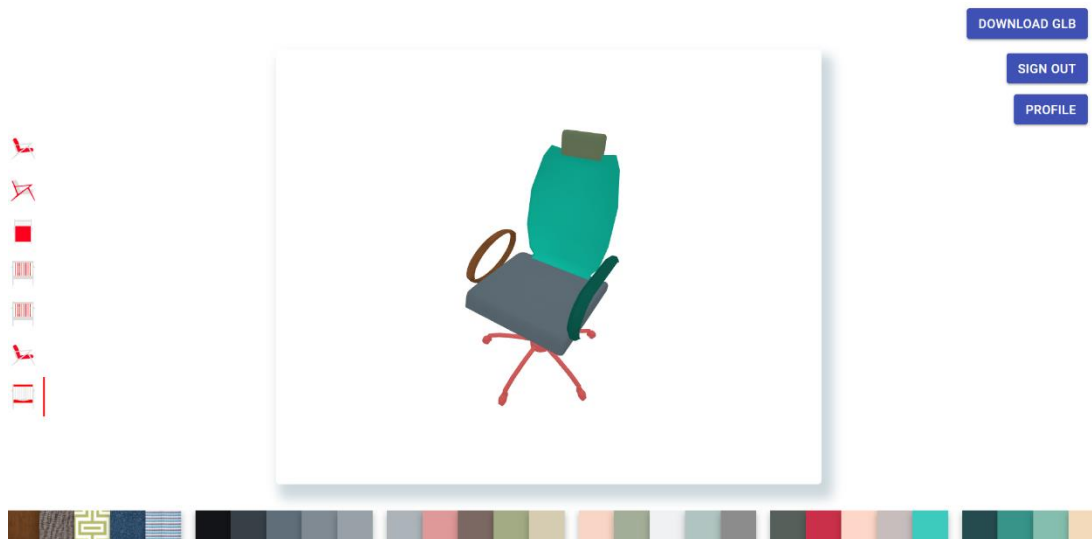
*Figure 8: File Structure and Names of 3D Configurator*



*Figure 9: UI components naming and placement*



*Figure 10: Example 1 of Custom Product*



*Figure 11: Example 2 of Custom Product*

### 5.2.2. Data Flow Diagram/Flowcharts:

A DFD represents how a system processes data and describes where the data comes from, where it goes and how the data is stored.

Some common DFD notations are as follows.

**Process** – Transforms incoming data flow into the outgoing data flow

**Data Store** – Represents the repositories of data in the system

**Data Flows** – Represents the pathway of data flow

Data is handled using a concept known as State Management.

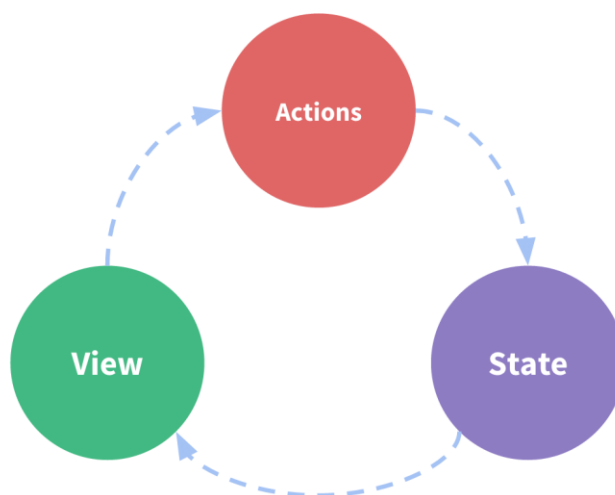
### 5.2.3. State Management Explained.

If you are building an app where components share data and are updated in response to UI interactions, the data on the interface is often referred to as state, it exists in memory and must be synced to the database. **Handling how that data is synced, shared, and updated** is what state management is about.

- The state is stored in the form of a JavaScript data structure. State describes the condition of the app at a specific point in time
- The **view**, a declarative description of the UI based on the current state
- The **actions**, the events that occur in the app based on user input, and trigger updates in the state

So,

- The UI is rendered based on that state
- When something happens (such as a user clicking a button), the state is updated based on what occurred
- The UI re-renders based on the new state



*Figure 12: State Management Overview*

React applications are built using components and they manage their state internally and it works well for applications with few components, but when the application grows bigger, the complexity of managing states shared across components becomes difficult.

**Redux** was created to resolve this issue. It provides a central store that holds all states of your application. Each component can access the stored state without sending it from one component to another.

A) Each 3D object has a texture and color state. There are 18 objects in the demo. This is an extract of how the state is declared.

```
//Colors and Textures State
const [colorSP, setColorSP] = useState(null);
const [textureSP, setTextureSP] = useState(null);
const [colorBK, setColorBK] = useState(null);
const [textureBK, setTextureBK] = useState(null);
const [colorBK1, setColorBK1] = useState(null);
const [textureBK1, setTextureBK1] = useState(null);
const [colorBK2, setColorBK2] = useState(null);
const [textureBK2, setTextureBK2] = useState(null);
const [colorBS, setColorBS] = useState(null);
const [textureBS, setTextureBS] = useState(null);
```

*Figure 13: Color and Texture States in Code*

B) The entire Scene Graph has objects removed and rendered onto it.

```
//Applied Object
const [appliedRightArm, setAppliedRightArm] = useState("RA1");
const [appliedLeftArm, setAppliedLeftArm] = useState("LA");
const [appliedBase, setAppliedBase] = useState("BS");
const [appliedBack, setAppliedBack] = useState("BK");
const [appliedHead, setAppliedHead] = useState("H");
const [appliedSeat, setAppliedSeat] = useState("ST");
const [appliedSupport, setAppliedSupport] = useState("SP");
```

*Figure 14:Scene Graph States in Code*

C) The Option component has an activeOption State to it

```
//currently used activeOption
const [activeOption, setActiveOption] = useState(0);
```

*Figure 15: Active Option State in Code*

D) The Download Button has state that trigger exporting of the Current Configuration

```
// For the GLTF Exporter to trigger rendering of full scene
const [clicked, setClicked] = useState(false);
const updateClick = (value) => {
  setClicked(value);
};
```

*Figure 16: Download button trigger State in Code*

This is how the trigger works.

```
<div className="download-btn">
  <DownloadBtn onClick={() => setClicked(true)} />
</div>
```

*Figure 17: Download Button onClick State Implementation*

#### 5.2.4. Interpreting the Product Configurator Workflow Design:

The **viewer** shows real-time updates from the UI

**Action:** User clicks on left arm **option**

Handler name: Set Active Option to Left Arm: Update Objects

Handler name: Set Active Option to Left Arm: Color or Texture

Handler description: UI dynamically switches object component to show possible left hands and set active option for color and texture manipulation to Left Arm

**Action:** User clicks on certain left arm **object**

Handler name: Set Applied Left Arm Object

Handler description: UI updates scene graph with that selected object

**Action:** User clicks on **color or texture**

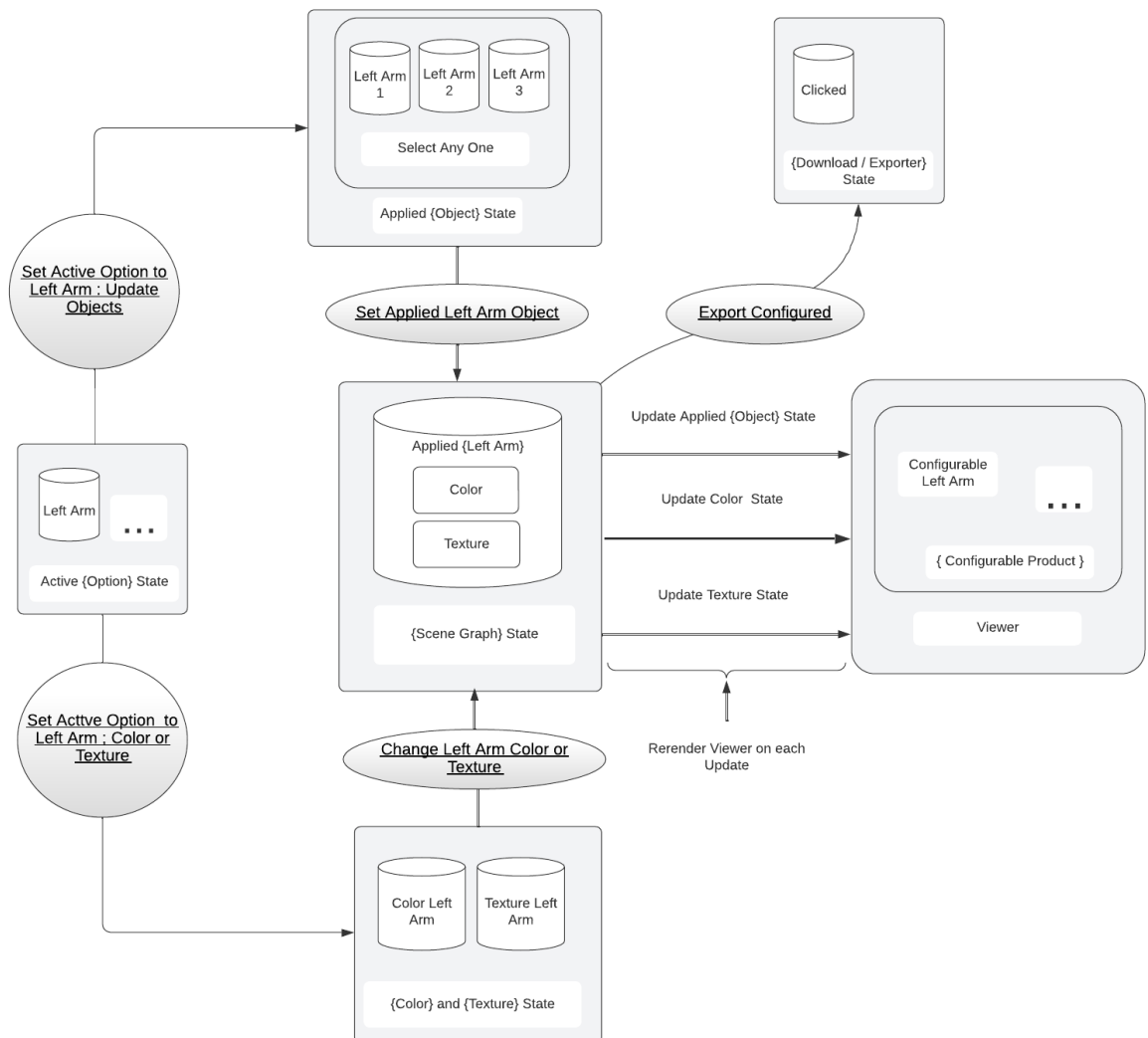
**Handler name:** Change Left Arm Color or Texture

**Handler description:** UI first checks what selected option is, then UI updates selected object's color or texture

**Action:** User clicks on **Download GLB**

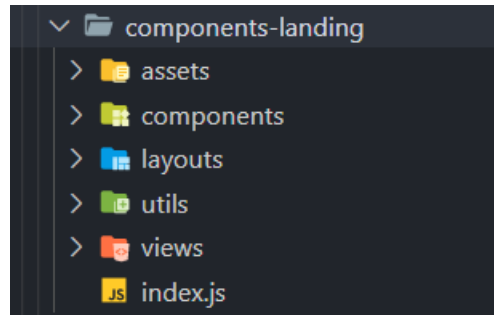
**Handler name:** Export Configured

**Handler description:** UI triggers Button state and exports the current configuration.

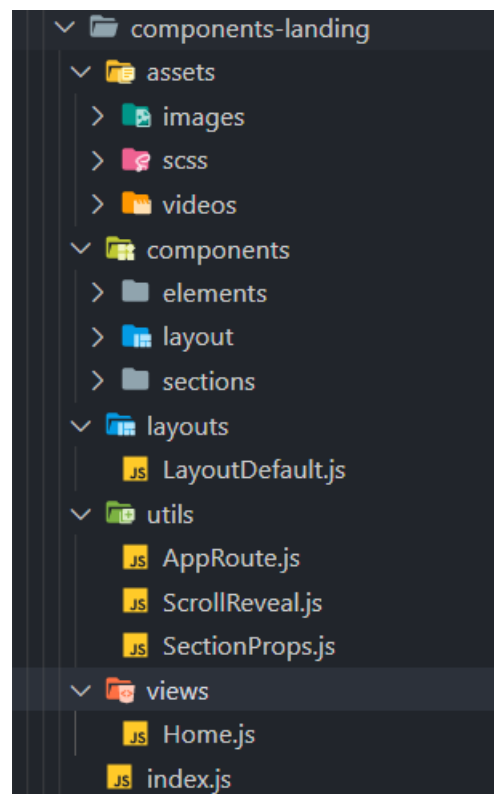


*Figure 18: 3D Product Configurator Workflow/DFD*

### 5.3. Landing Page



*Figure 19: Landing Page component Structure*



*Figure 20: Landing Page Components File Structure*

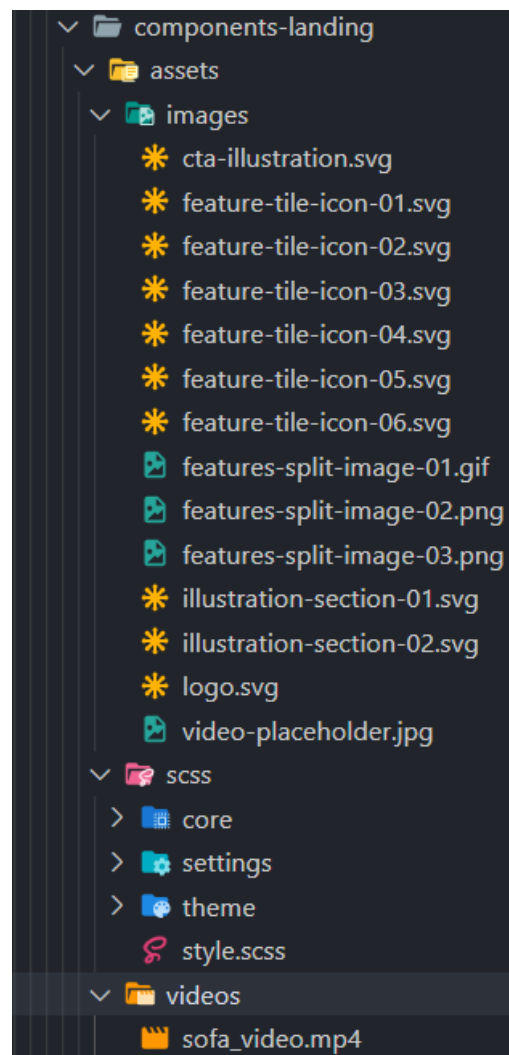
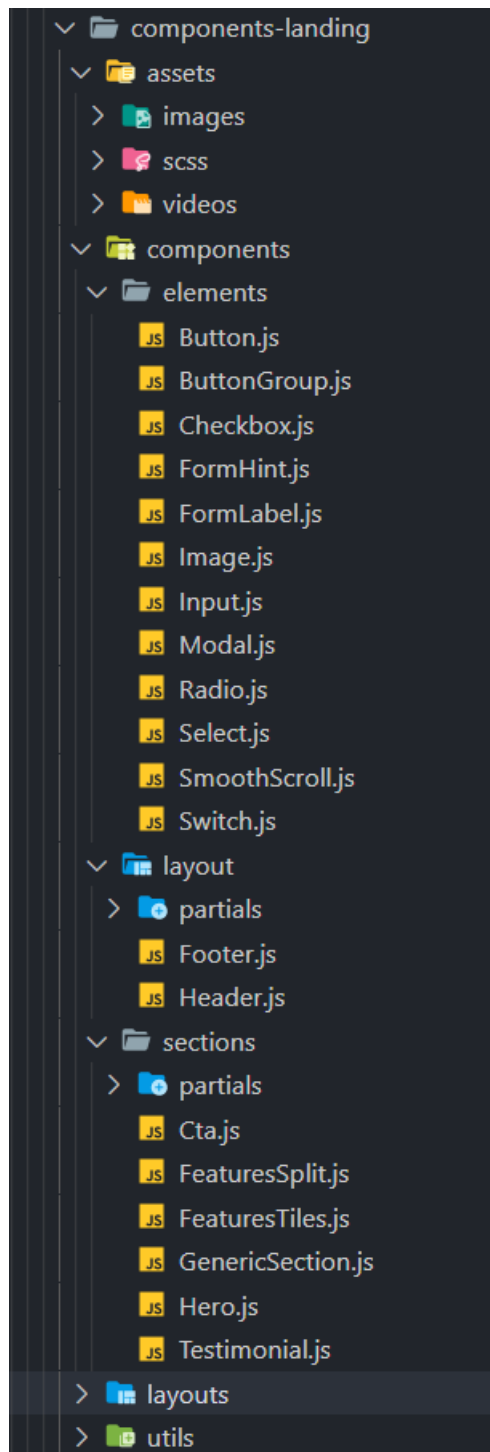
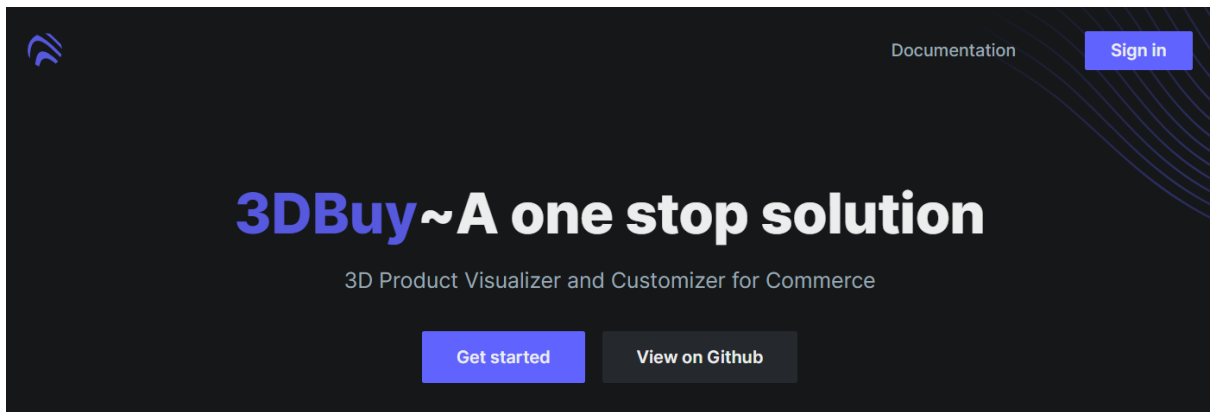


Figure 21: Landing Page Assets Management File Structuring

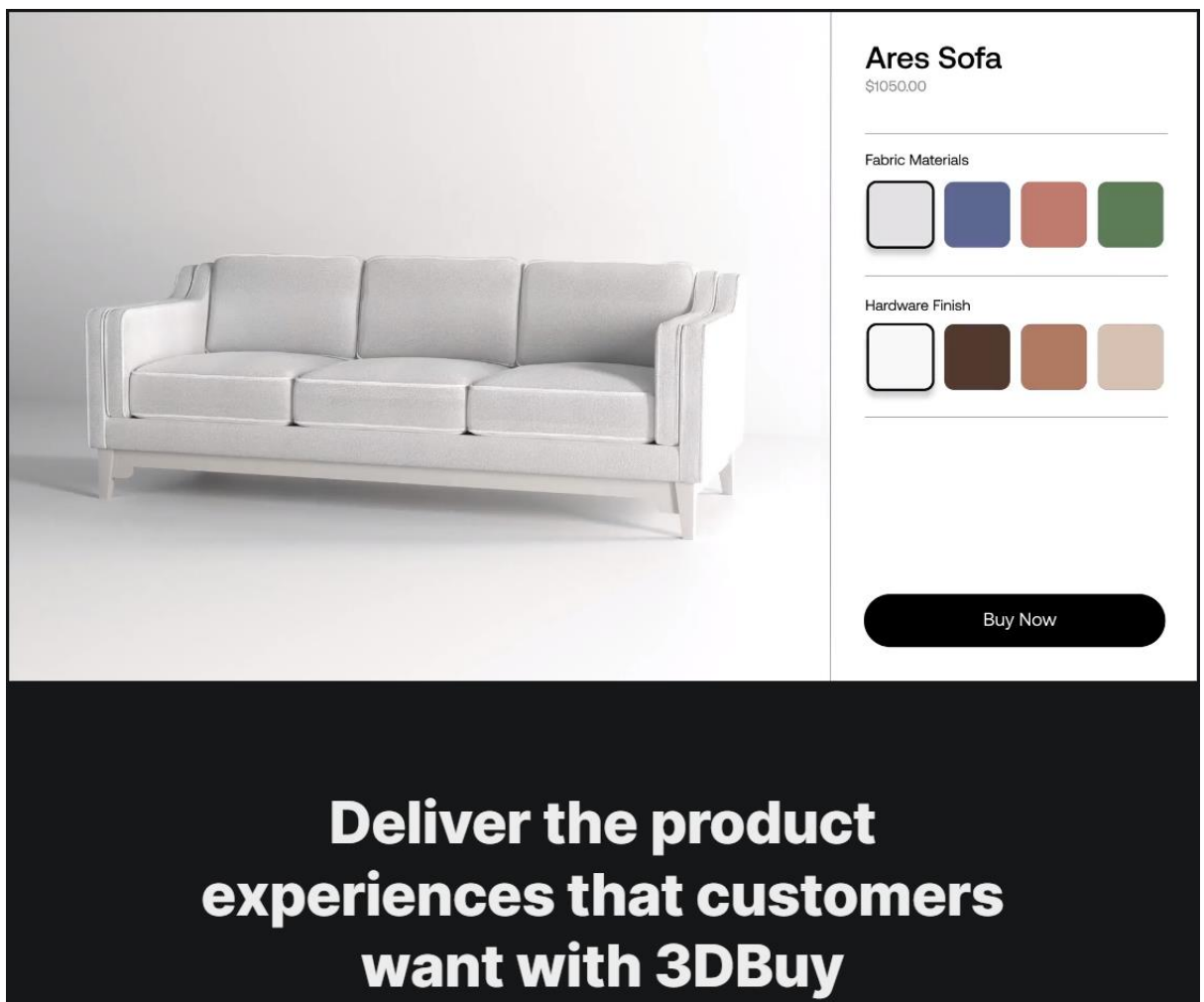




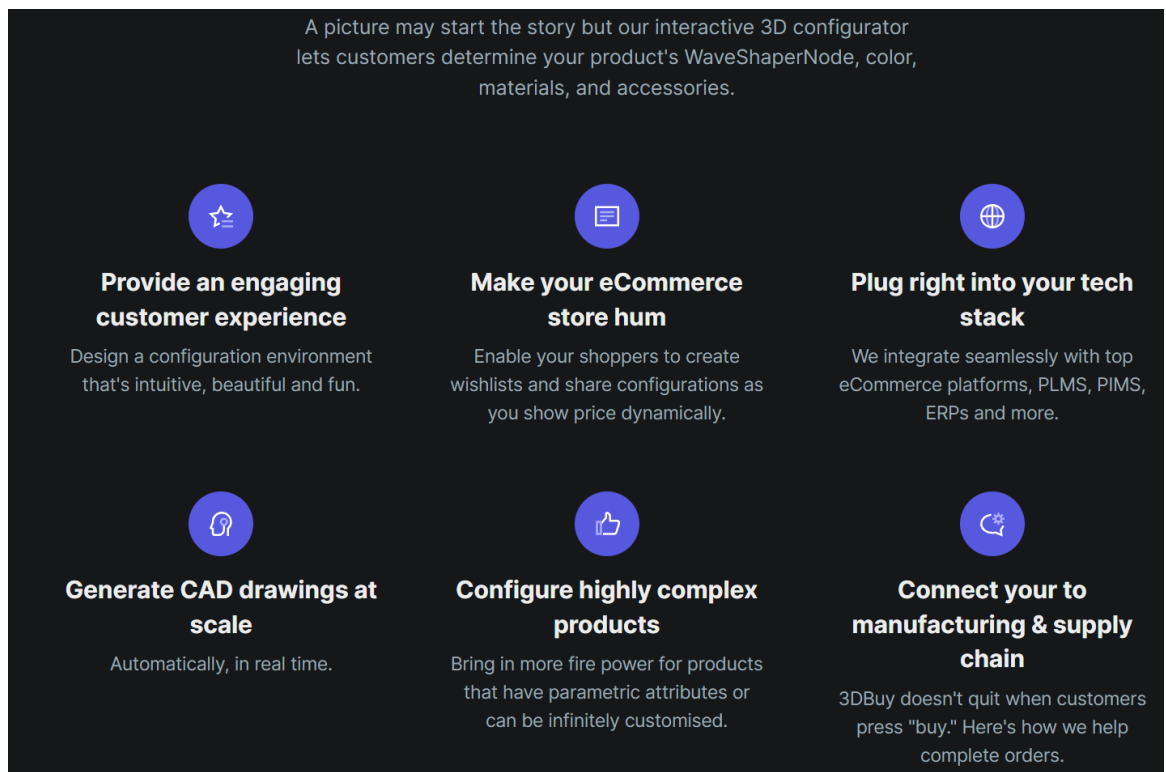
*Figure 22: Landing Page Layouts, Sections and Elements Structuring*



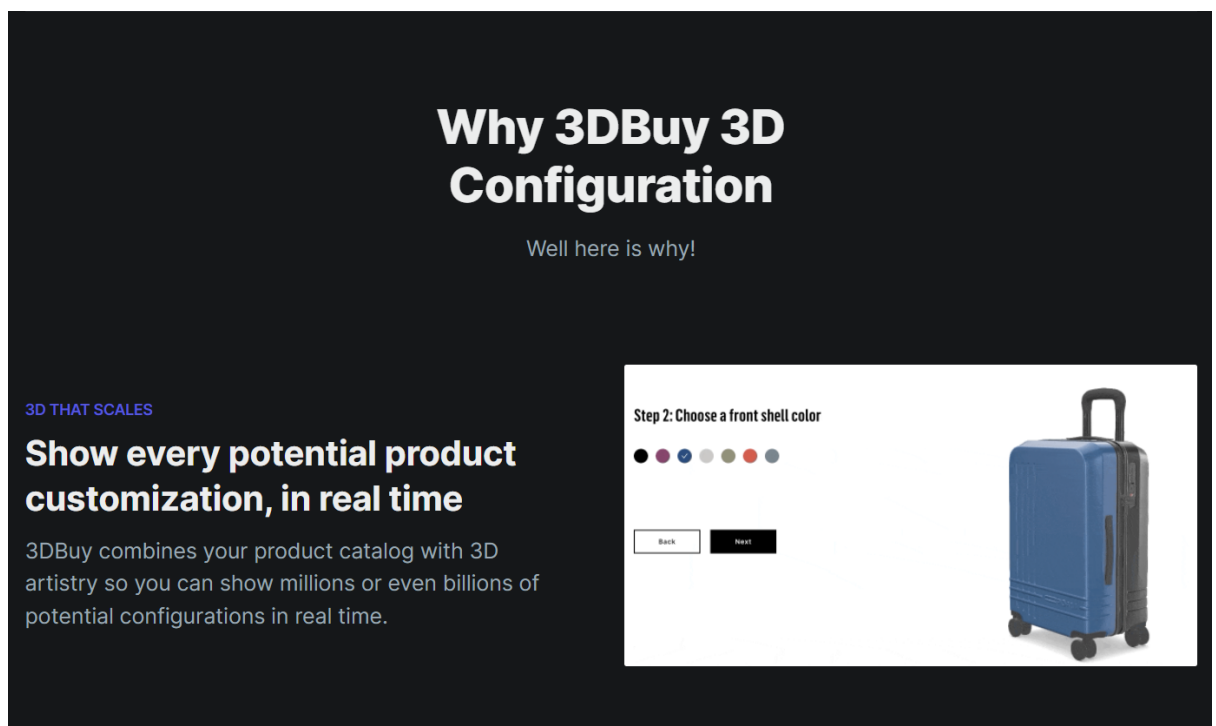
*Figure 23: Landing Page Header and Hero Section 1*



*Figure 24: Landing Page Hero Section 2*



*Figure 25: Landing Page Features Tile Section*

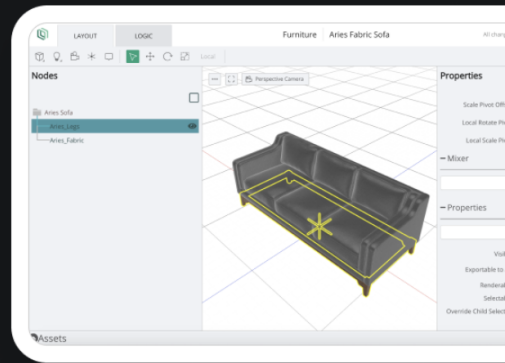


*Figure 26: Landing Page Features Split Section 1*

#### TEAM EFFICIENCY

## Make your team better while making your product experience better

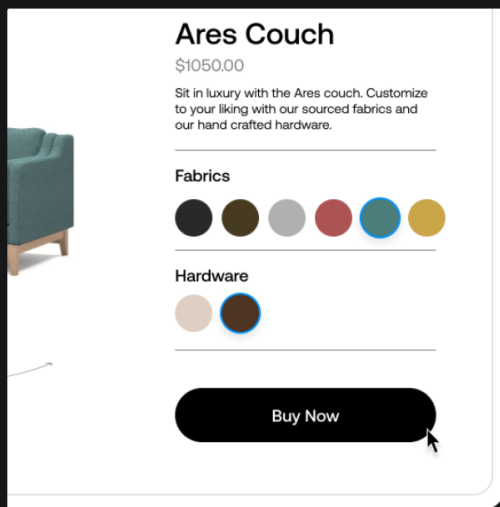
From 3D asset management to a code-free product catalog – Threekit makes it easy to collaborate across teams for a great result.



## What customers have to say

Just in case you were wondering.

*Figure 27: Landing Page Features Split Section 2*

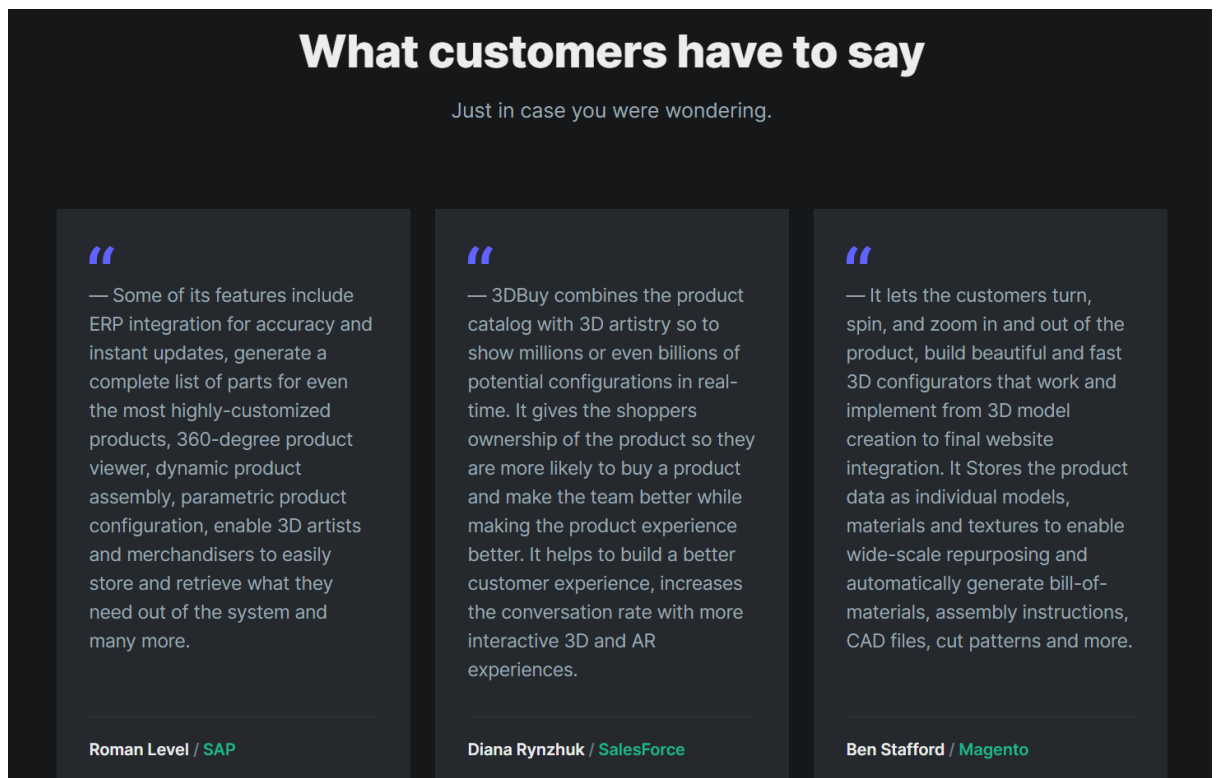


#### BUYER ENGAGEMENT

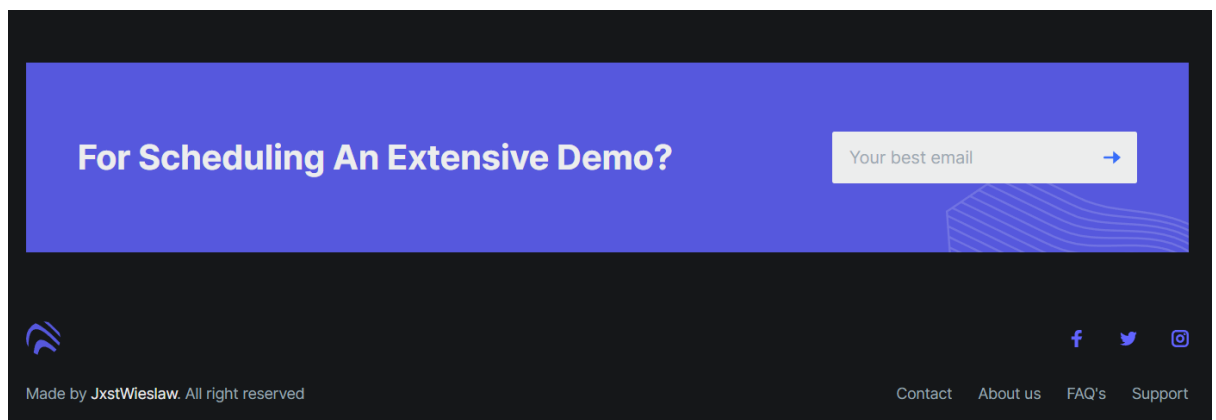
## Give your shoppers ownership of the product so they'll press "buy"

When shoppers use a 3D configurator, they are 20% more likely to buy a product. It gets them involved in the process and allows them to make it their own.

*Figure 28: Landing Page Features Split Section 3*



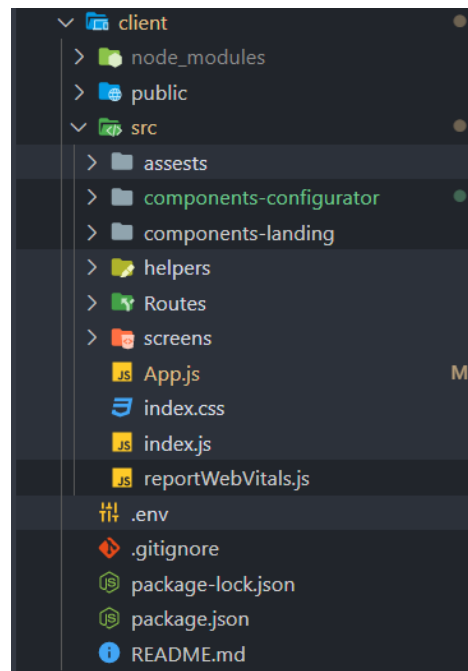
*Figure 29: Landing Page Testimonials Section*



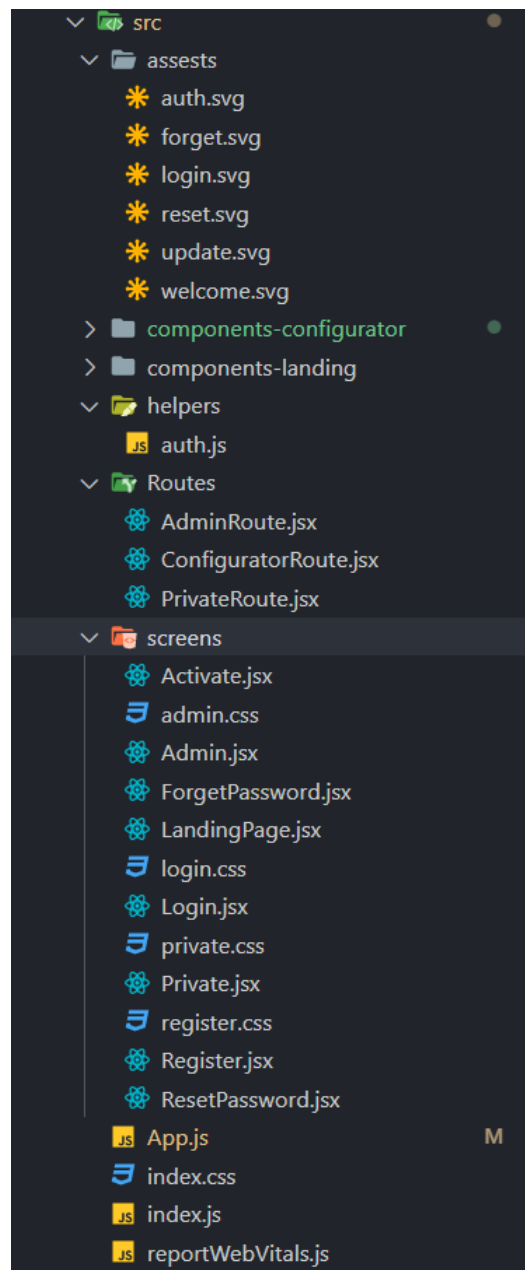
*Figure 30: Landing Page Footer and Cta Section*

## 5.4. Authentication and Authorization app

### 5.4.1. Auth App Front End Folder Structure

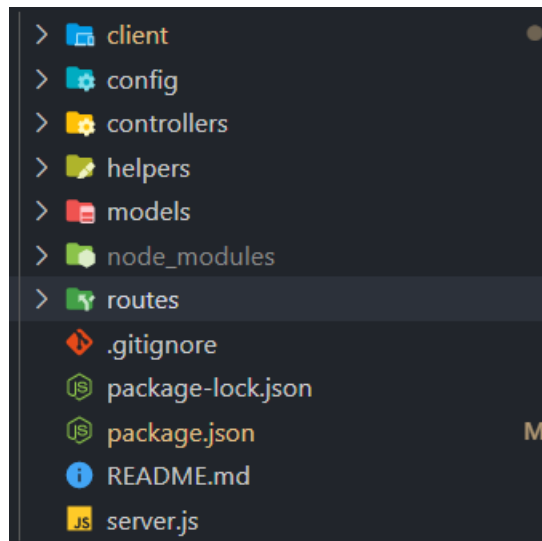


*Figure 31: Auth App React Components Structure: Highlighted*

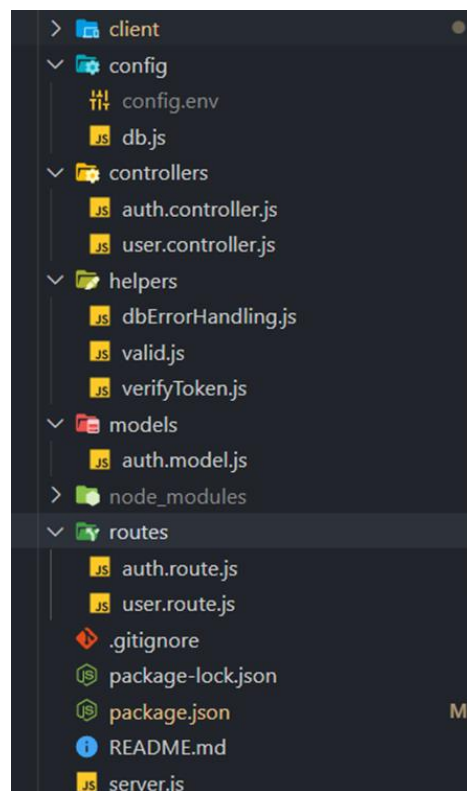


*Figure 32: Auth App React Components File Structuring: Expanded Folders*

### 5.4.2. Auth App Server/Back End Folder Structure



*Figure 33: Auth App ExpressJS Base Folder Structuring*

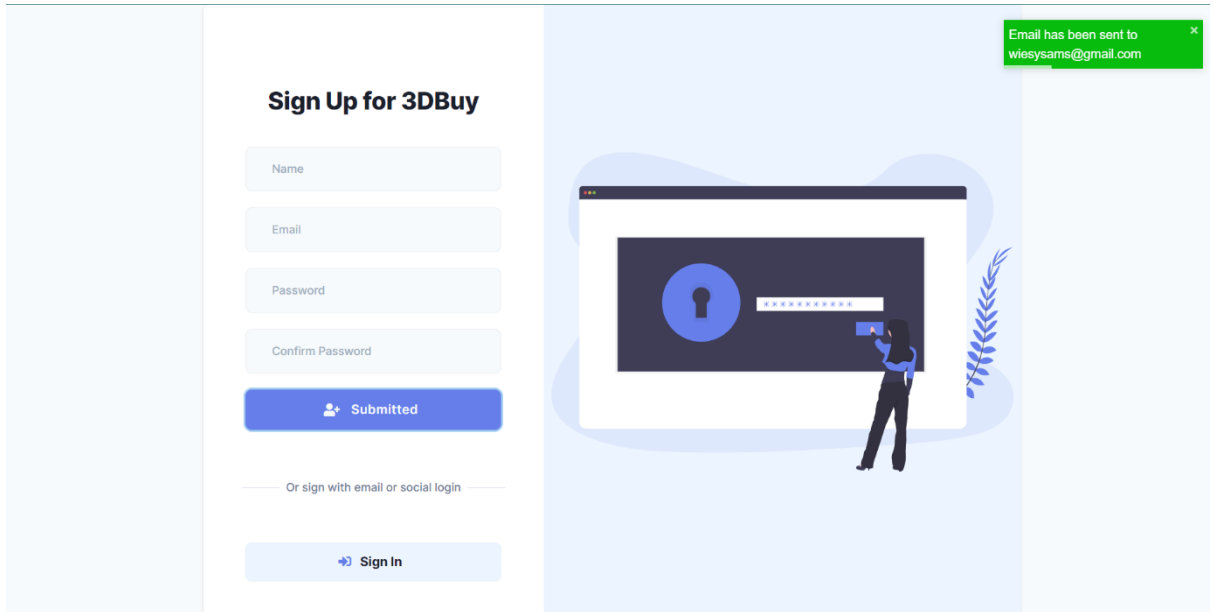


*Figure 34: Auth App Express JS Server File Structuring*



### 5.4.3. Sign up

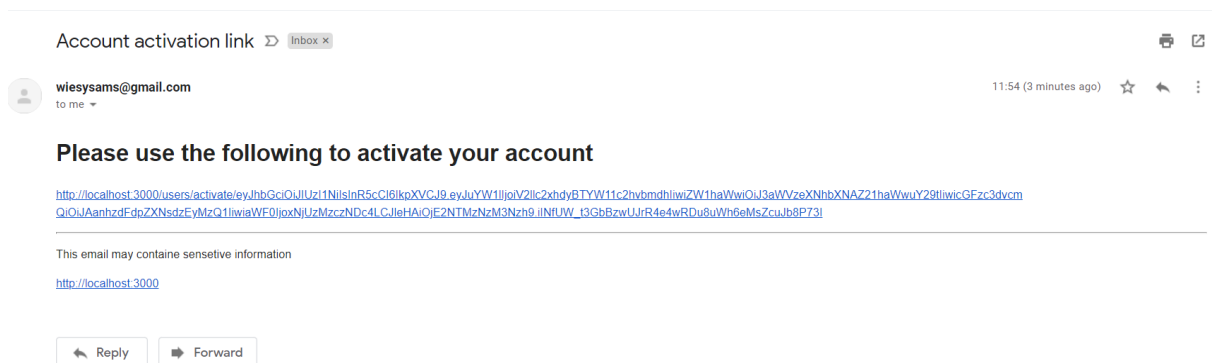
User signing up with credentials. Email sent after signing up for account activation.



The image shows a web form titled "Sign Up for 3DBuy". The form has four input fields: "Name", "Email", "Password", and "Confirm Password". Below these fields is a blue button labeled "Submitted" with a user icon. Underneath the button is a link that says "Or sign with email or social login". At the bottom of the form is a light blue button labeled "Sign In" with a right-pointing arrow. To the right of the form is a large illustration of a person standing next to a computer monitor. The monitor displays a login screen with a keyhole icon and a password field. A green notification box in the top right corner of the page states "Email has been sent to wiesysams@gmail.com".

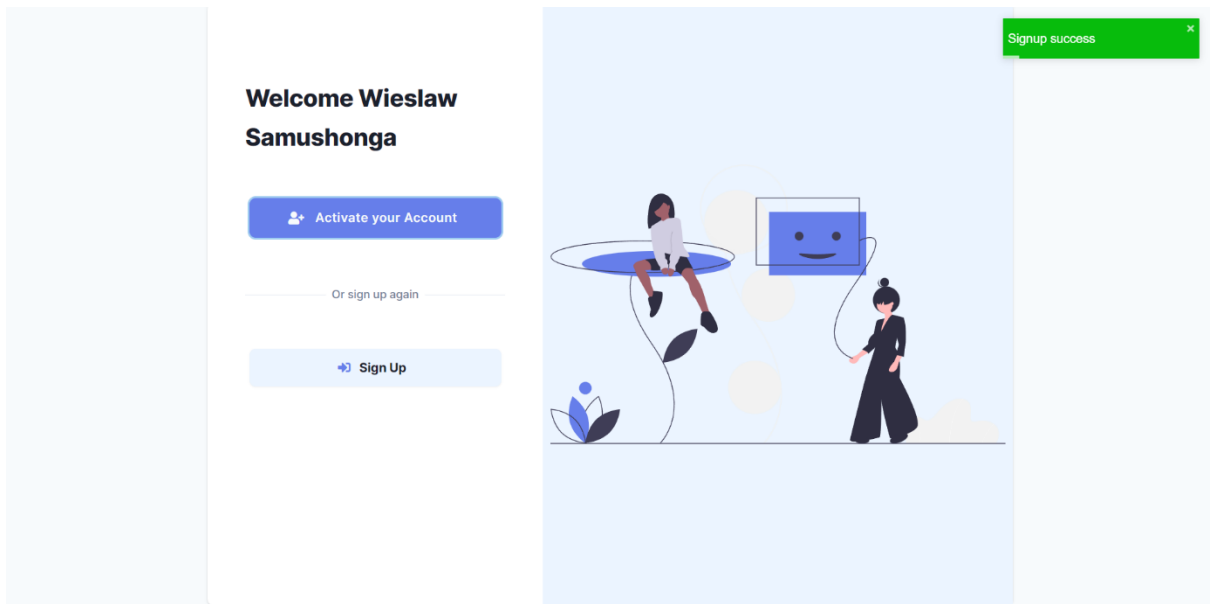
*Figure 35: Sign Up Form with email sent notification*

### 5.4.4. Account Activation Email with Link



*Figure 36: Email with Account Activation Link*

### 5.4.5. Activate Account Page

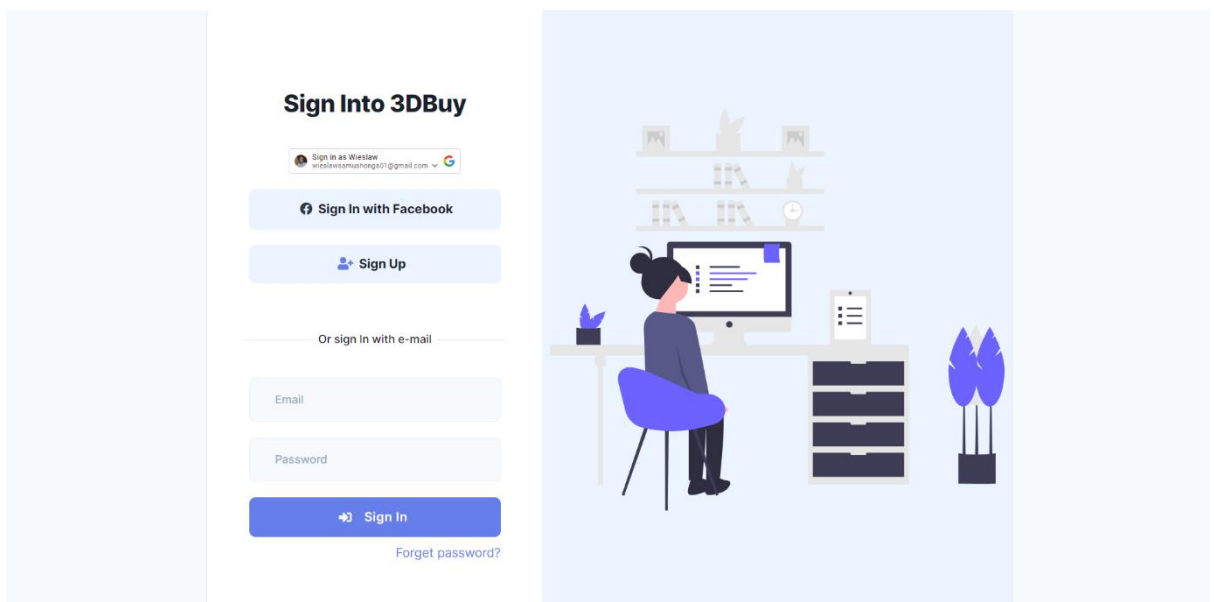


*Figure 37: Account Activation Page*

#### 5.4.6. Sign In

For Social Login – no signup page required, no email sent for verification as already verified by domain hence direct sign up and logging in.

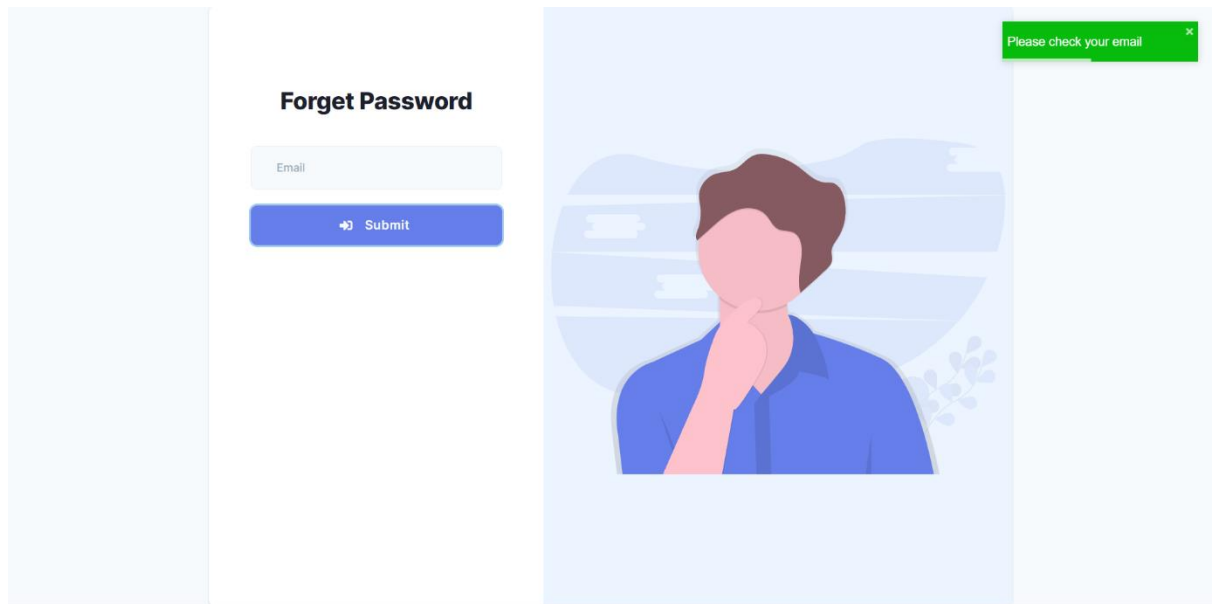
For Email Login –Key in your registered email and corresponding password



*Figure 38: Sign in Page*

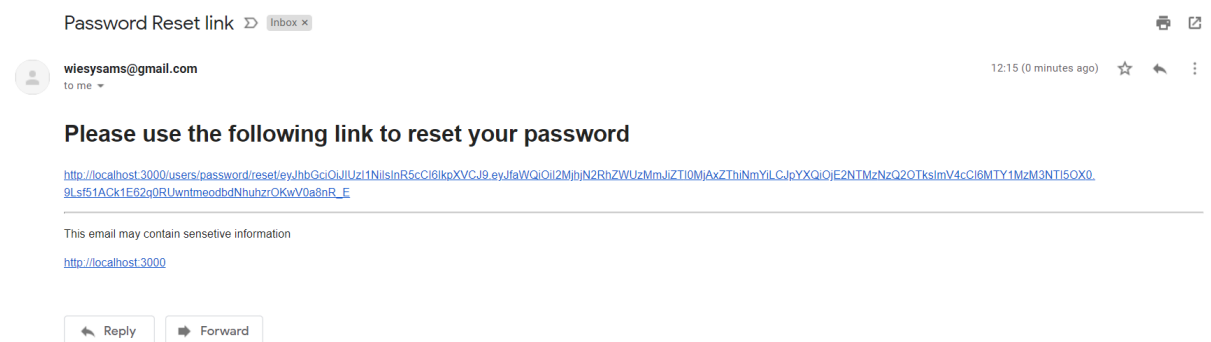
#### 5.4.7. Forgot Password Page.

Email with Reset Password Form Link sent to account



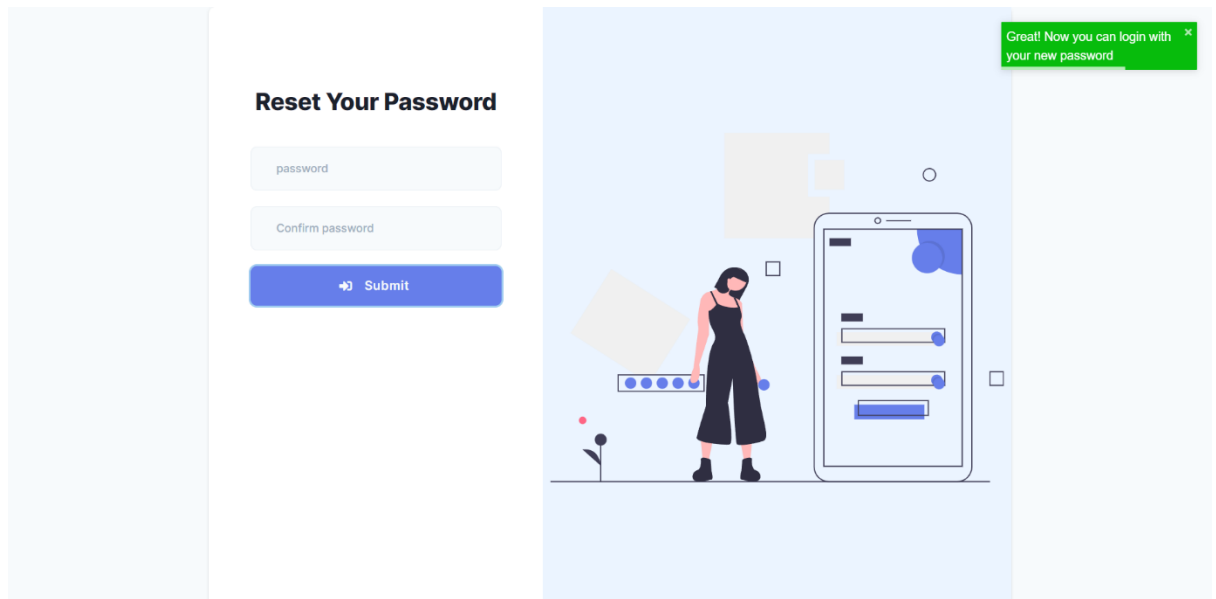
*Figure 39: Forget Password Page*

### 5.4.8. Email with Password Reset Link



*Figure 40: Password Reset Link Email*

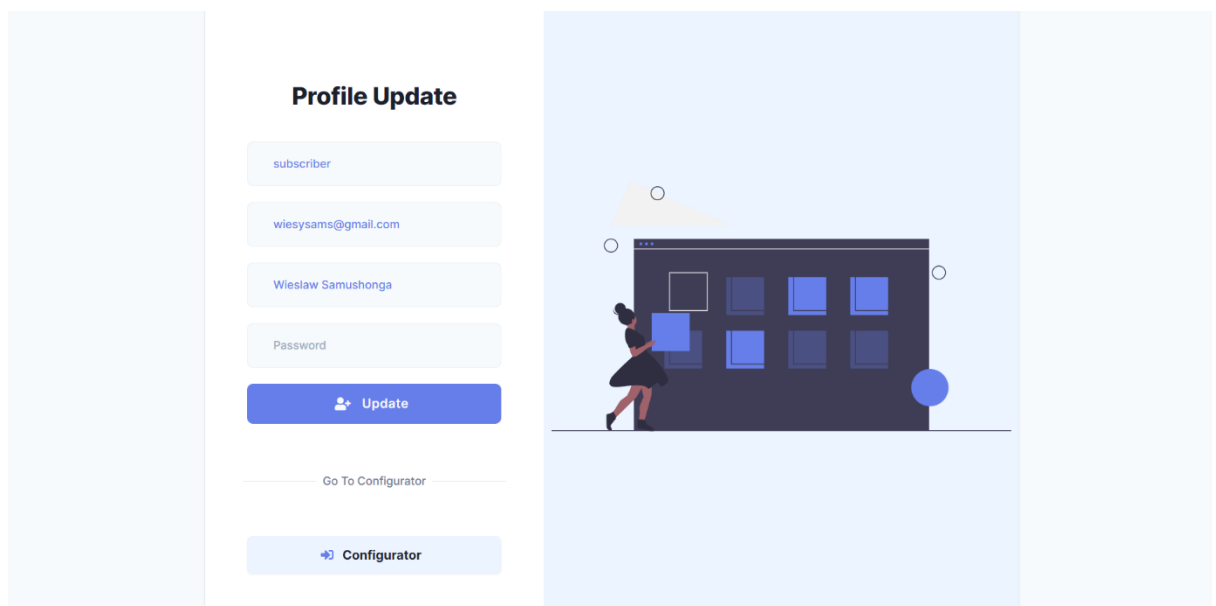
### 5.4.9. Password Reset Page



*Figure 41: Password Reset Page*

### 5.4.10. Private Profile Page

When successfully logged in, you can view the corresponding profile page



*Figure 42: Profile Update Page*

## 5.5. Code Snippets

### 5.5.1. SignUp Form Submit

```
const handleSubmit = e => {
  e.preventDefault();
  if (name && email && password1) {
    if (password1 === password2) {
      setFormData({ ...formData, textChange: 'Submitting' });
      axios
        .post(`${process.env.REACT_APP_API_URL}/register`, {
          name,
          email,
          password: password1
        })
        .then(res => {
          setFormData({
            ...formData,
            name: '',
            email: '',
            password1: '',
            password2: '',
            textChange: 'Submitted'
          });

          toast.success(res.data.message);
        })
        .catch(err => {
          setFormData({
            ...formData,
            name: '',
            email: '',
            password1: '',
            password2: '',
            textChange: 'Sign Up'
          });
          console.log(err.response);
          toast.error(err.response.data.errors);
        });
    } else {
      toast.error("Passwords don't matches");
    }
  } else {
    toast.error('Please fill all fields');
  }
};

export default Register;
```

*Figure 43: Sign Up Form Code Snippet*

### 5.5.2. Account Activation Page: Activate

```
const Activate = ({ match }) => {
  const [formData, setFormData] = useState({
    name: '',
    token: '',
    show: true
  });

  useEffect(() => {
    let token = match.params.token;
    let { name } = jwt.decode(token);

    if (token) {
      setFormData({ ...formData, name, token });
    }

    console.log(token, name);
  }, [match.params]);
  const { name, token, show } = formData;

  const handleSubmit = e => {
    e.preventDefault();

    axios
      .post(`${process.env.REACT_APP_API_URL}/activation`, {
        token
      })
      .then(res => {
        setFormData({
          ...formData,
          show: false
        });

        toast.success(res.data.message);
      })
      .catch(err => {
        toast.error(err.response.data.errors);
      });
  };

  return (
    <>
    {From Design Here}
    </>
  );
};

export default Activate;
```

*Figure 44: Account Activation Page Code Snippet*

### 5.5.3. Sign in Page : Google Login

```
const Login = ({ history }) => {
  const [formData, setFormData] = useState({
    email: "",
    password1: "",
    textChange: "Sign In",
  });
  const { email, password1, textChange } = formData;
  const handleChange = (text) => (e) => {
    setFormData({ ...formData, [text]: e.target.value });
  };

  const sendGoogleToken = (tokenId) => {
    console.log(tokenId);
    axios
      .post(`${process.env.REACT_APP_API_URL}/googlelogin`, {
        idToken: tokenId,
      })
      .then((res) => {
        console.log(res.data);
        informParent(res);
      })
      .catch((error) => {
        console.log("GOOGLE SIGNIN ERROR", error.response);
      });
  };

  const informParent = (response) => {
    authenticate(response, () => {
      console.log("here");
      isAuth() && history.push("/configurator");
    });
  };

  return (
    <div className="w-full flex-1 mt-8 text-indigo-500">
      <div className="flex flex-col items-center">
        <GoogleLogin
          onSuccess={responseGoogle}
          onError={() => {
            console.log("Login Failed");
          }}
        />
      </div>
    </div>
  );
};

export default Login;
```

*Figure 45: Sign in Page Code Snippet*

### 5.5.4. Forgot Password Page: Submit

```
const ForgetPassword = ({history}) => {
  const [formData, setFormData] = useState({
    email: '',
    textChange: 'Submit'
  });
  const { email, textChange } = formData;
  const handleChange = text => e => {
    setFormData({ ...formData, [text]: e.target.value });
  };
  const handleSubmit = e => {
    e.preventDefault();
    if (email) {
      setFormData({ ...formData, textChange: 'Submitting' });
      axios
        .put(`${process.env.REACT_APP_API_URL}/forgotpassword`, {
          email
        })
        .then(res => {

          setFormData({
            ...formData,
            email: '',
          });
          toast.success('Please check your email');

        })
        .catch(err => {
          console.log(err.response)
          toast.error(err.response.data.error);
        });
    } else {
      toast.error('Please fill all fields');
    }
  };
  return (
    <div className='min-h-screen bg-gray-100 text-gray-900 flex justify-center'>
      {Enter Page Design Here}
    </div>
  );
};

export default ForgetPassword;
```

*Figure 46: Forget Password Page Code Snippet*



### 5.5.5. Password Reset Page: Submit

```
const ResetPassword = ({match}) => {
  const [formData, setFormData] = useState({
    password1: '',
    password2: '',
    token: '',
    textChange: 'Submit'
  });

  const handleSubmit = e => {
    console.log(password1, password2)
    e.preventDefault();
    if ((password1 === password2) && password1 && password2) {
      setFormData({ ...formData, textChange: 'Submitting' });
      axios
        .put(`${process.env.REACT_APP_API_URL}/resetpassword`, {
          newPassword: password1,
          resetPasswordLink: token
        })
        .then(res => {
          console.log(res.data.message)
          setFormData({
            ...formData,
            password1: '',
            password2: ''
          });
          toast.success(res.data.message);
        })
        .catch(err => {
          toast.error('Something is wrong try again');
        });
    } else {
      toast.error('Passwords don\'t matches');
    }
  };
  return (
    <div className='min-h-screen bg-gray-100 text-gray-900 flex justify-center'>
      {Form Design Here}
    </div>
  );
};

export default ResetPassword;
```

*Figure 47: Password Reset Page Code Snippet*

### 5.5.6. Profile Update Page: Load Profile

```
const Private = ({ history }) => {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    password1: '',
    textChange: 'Update',
    role: ''
  });

  useEffect(() => {
    loadProfile();
  }, []);

  const loadProfile = () => {
    const token = getCookie('token');
    axios
      .get(`${process.env.REACT_APP_API_URL}/user/${isAuth()._id}`, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      })
      .then(res => {
        const { role, name, email } = res.data;
        setFormData({ ...formData, role, name, email });
      })
      .catch(err => {
        toast.error(`Error To Your Information ${err.response.statusText}`);
        if (err.response.status === 401) {
          signout(() => {
            history.push('/login');
          });
        }
      });
  });

  const { name, email, password1, textChange, role } = formData;
  const handleChange = text => e => {
    setFormData({ ...formData, [text]: e.target.value });
  };

  return (
    <div className='min-h-screen bg-gray-100 text-gray-900 flex justify-center'>
      {Enter Page Design Here}
    </div>
  );
};

export default Private;
```

*Figure 48: Profile Update Page Code Snippet*

## CHAPTER 6

### Conclusion

The project exposed me to the latest technologies in 3D web development. The project successfully demonstrated that development is a game of critical thinking and constant analytics.

The main objective of the project was **to avail a product** which would **increase the turnover** of a business by **improved customer experiences**. An enhanced visual experience was achieved and as such a business can:

- Sell more by letting customers configure complex products, in real time
- Show every potential product customization, in real time
- Make 3D asset management easier
- Configure highly complex products
- Let customers create products unique to them
- Enter the Metaverse with product NFTs
- Integration into any website with a built-in user interface

### 6.1. Future Scope

- Click on the Product and automatically select the component to be People should be able to click on a model to make changes. Personalization shouldn't be a dropdown menu or via an active option tile.
- Saves Changes. Customers can save details about their customized products for easy access later.
- Supports Dynamic Pricing. Configurator shows exact pricing, and pricing changes, as customer customizes products.
- Import 3D or CAD files. If you don't have those, 3DBuy will build a 3D file from representative photos and materials you send over.
- Easier integrations and configuration parameters buildups.

## REFERENCES

- [1] Chu, C. H., Cheng, C. Y. and Wu, C. W., 2005, "Applications of the web-based collaborative visualization in distributed product development," *Computers in Industry*, under review.
- [2] Fagerstrom, B. and Jackson, M., 2002, "Efficient collaboration between main and sub-suppliers," *Computers in Industry*, Vol. 49, pp. 25-35. 6. Fuxin, F. and Edlund, S., 2001, "Categorisation of geometry users," *Concurrent Engineering: Research and Application*, Vol. 9, No. 1, pp. 15-22.
- [3] Helander, M. G. and Jiao, J., 2002, "Research on e-Product development (ePD) for mass customization," *Technovation*, Vol. 22, No. 11, pp. 717-724.
- [4] <https://www.soft8soft.com/do-it-yourself-3d-product-configurator/>
- [5] <https://www.konfigear.com/>
- [6] <https://www.designnbuy.com/product-configurator.html>
- [7] <https://hapticmedia.com/blog/online-3d-product-configurator-for-ecommerce/>
- [8] <https://configureid.com/2021/09/07/the-ultimate-guide-to-finding-the-perfect-3d-product-configurator/>
- [9] <https://threejs.org/>
- [10] <https://reactjs.org/>
- [11] <https://expressjs.com/>
- [12] <https://docs.pmnd.rs/>
- [13] <https://www.npmjs.com/>
- [14] <https://www.mongodb.com/>
- [15] <https://nodejs.org/en/>
- [16] Li, W. D., Lu, W. F., Fuh, J. Y. H. and Wong, Y. S., 2005, "Collaborative computer-aided design: Research and development status," *Computer-Aided Design*, Vol. 37, No. 9, pp. 931-940.
- [17] Roy, U. and Kodkani, S. S., 1999, "Product modeling within the framework of the World Wide Web," *IIE Trans*, Vol. 31, pp. 667-677.
- [18] Shi, Z. N., 2004, "The feasibility study and implementation of ITRI SME-PDM on Microsoft .NET platform," Department of Computer Science and Information Engineering, National Taiwan University, Master Thesis.
- [19] Shyamsundar, N. and Gadh, R., 2002, "Collaborative virtual prototyping of product assemblies over the Internet," *Computer-Aided Design*, Vol. 34, No. 10, pp. 755-769.
- [20] Ulrich, K. T. and Eppinger, S. D., 2004, *Product Design and Development*, McGraw Hill, 3rd Edition.