



FEDERAL UNIVERSITY OF SANTA CATARINA  
TECHNOLOGY CENTER  
AUTOMATION AND SYSTEMS DEPARTMENT  
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Gustavo Pacheco Figueiredo Pereira

**Web Based Application for Precision Glass Molding**

Aachen, Germany  
2024

Gustavo Pacheco Figueiredo Pereira

## **Web Based Application for Precision Glass Molding**

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.  
Supervisor: Prof. Marcelo Ricardo Stemmer, Dr.  
Co-supervisor: Cheng Jiang, M.Sc. M.Eng

Aachen, Germany  
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.  
Dados inseridos pelo próprio autor.

Pacheco Figueiredo Pereira, Gustavo  
Web Based Application for Precision Glass Molding /  
Gustavo Pacheco Figueiredo Pereira ; orientadora, Marcelo  
Ricardo Stemmer, coorientadora, Cheng Jiang, 2024.  
98 p.

Trabalho de Conclusão de Curso (graduação) -  
Universidade Federal de Santa Catarina, Centro Tecnológico,  
Graduação em Engenharia de Controle e Automação,  
Florianópolis, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2.  
Desenvolvimento web. 3. Moldagem de Lentes de Precisão. I.  
Ricardo Stemmer, Marcelo . II. Jiang, Cheng. III.  
Universidade Federal de Santa Catarina. Graduação em  
Engenharia de Controle e Automação. IV. Título.

Gustavo Pacheco Figueiredo Pereira

**Web Based Application for Precision Glass Molding**

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, June 30, 2024.

Prof. Marcelo De Lellis Costa De Oliveira, Dr.  
Course Coordinator

**Examining Board:**

Prof. Marcelo Ricardo Stemmer, Dr.  
Advisor  
UFSC/CTC/DAS

Cheng Jiang, M.Sc. M.Eng.  
Supervisor  
Fraunhofer Institute for Production Technology IPT.

João Paulo Zomer Machado,  
Evaluator  
UFSC/CTC/Labmetro

Prof. Prof. Eduardo Camponogara, Dr.  
Board President  
UFSC/CTC/DAS

## **ACKNOWLEDGEMENTS**

First, I would like to express my deepest gratitude to my family, my father Saulo Figueiredo Pereira, my mother Márcia Regina Pacheco Pereira, and my sister Júlia Pacheco Figueiredo Pereira. Thank you my close family, for their persistent support and encouragement throughout my academic and professional journey. I'm also grateful for my friends for their support, advises, laughs, hearing my concerns, and encouraging me during this process. I also want to thank the BRAACHEN program and Fraunhofer Institute for Production Technology IPT for the experience and knowledge that I have acquired during this year in Germany. Especially to Cheng Jiang, my local supervisor, for his guidance, expertise, mentorship, and valuable lessons throughout the development of this project. To professor Marcelo Ricardo Stemmer, my academic advisor, thank you for the valuable insights and feedback provided during my entire internship.

## DISCLAIMER

Aachen, June 28th, 2024.

As representative of the Fraunhofer Institute for Production Technology IPT in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property, that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation (*“Trabalho de Conclusão de Curso”*), in accordance with the *Resolução Normativa n° 126/2019/CUn*.

**Cheng  
Jiang**



Digitally signed by  
Cheng Jiang  
Date: 2024.12.02  
09:19:41 +01'00'

---

Cheng Jiang

Fraunhofer Institute for Production Technology IPT

## ABSTRACT

The manufacturing industry is increasingly shifting towards advanced computational techniques like finite element analysis for process optimization and quality control. Precision glass molding is a highly efficient manufacturing process used to produce complex-shaped glass optics. This complex geometries and materials, requires robust simulation capabilities to ensure precision and efficiency. To address this challenge, researchers at Fraunhofer IPT are developing an integrated numerical simulation software. This application will provide an intuitive interface for creating glass molding simulations, automating certain tasks, and summarizing the results without prior simulation experience. It features three key modules - a Database Generator, Simulation Generator, and Simulation Visualizer. The focus of the current project is on enhancing the Database Generator module, which serves as the initial step for users to generate all the necessary data for subsequent simulation and analysis.

**Keywords:** MongoDB. Non-relational Database. CRUD application. Manufacturing Process.

## RESUMO

A indústria de manufatura está cada vez mais se voltando para técnicas computacionais avançadas, como a análise de elementos finitos, para otimização de processos e controle de qualidade. A moldagem de vidro de precisão é um processo de fabricação altamente eficiente utilizado para produzir ópticas de vidro com formas complexas. Essas geometrias complexas e materiais exigem capacidades robustas de simulação para garantir precisão e eficiência. Para solucionar esse desafio, pesquisadores do Instituto Fraunhofer IPT estão desenvolvendo uma aplicação integrada a um software de simulação numérica. Essa aplicação fornecerá uma interface intuitiva para a criação de simulações de moldagem de vidro, automatizando certas tarefas e levantamento de resultados, sem a necessidade de experiência prévia em simulação. Ela conta com três módulos principais - um Gerador de Banco de Dados, Gerador de Simulação e Visualizador de Simulação. O foco do projeto atual está em aprimorar o módulo Gerador de Banco de Dados, que serve como o passo inicial para os usuários gerarem todos os dados necessários para simulações e análises subsequentes.

**Palavras-chave:** MongoDB. Banco de Dados Não Relacional. Aplicação CRUD. Processos de Manufatura.



## LIST OF FIGURES

Figure 1 – Use Case Diagram with the main functional requirements of the software . . . . .	15
Figure 2 – SIMPGM Old Version Interface <sup>1</sup> . . . . .	16
Figure 3 – Simulation Visualizer Module First Version Interface <sup>1</sup> . . . . .	17
Figure 4 – Precision Glass Molding Process . . . . .	20
Figure 5 – JWT token server and User side . . . . .	24
Figure 6 – Multitenant Architecture . . . . .	29
Figure 7 – Package Diagram Backend Structure . . . . .	35
Figure 8 – JWT validation logic . . . . .	36
Figure 9 – Frontend Directory Structure . . . . .	38
Figure 10 – Login Form <sup>1</sup> . . . . .	39
Figure 11 – Definition Form Implementation Result <sup>1</sup> . . . . .	45
Figure 12 – UI screenshot with two creation options for the user <sup>1</sup> . . . . .	46
Figure 13 – UI screenshot with Project and Machine Generators . . . . .	47
Figure 14 – Create Option flow chart . . . . .	48
Figure 15 – UI from Project and Machine generators <sup>1</sup> . . . . .	49
Figure 16 – UI from Coating and Process Parameters generators. <sup>1</sup> . . . . .	50
Figure 17 – PDF Visualizer displaying a PDF . . . . .	52
Figure 18 – PDF Visualizer Logic flow chart . . . . .	53
Figure 19 – API diagram with the Request/Response with the previous version approach for the Drop-down data . . . . .	54
Figure 20 – API diagram with the Request/Response with the new version approach for the Drop-down data . . . . .	55
Figure 21 – UI from Cooling Plate generator <sup>1</sup> . . . . .	56
Figure 22 – UI from Sleeve generator <sup>1</sup> . . . . .	56
Figure 23 – UI from Holder generator <sup>1</sup> . . . . .	57
Figure 24 – API diagram with the Request/Response with the plot generation logic . . . . .	58
Figure 25 – Preview Plot Flowchart . . . . .	59
Figure 26 – Glass Preform UI with Meniscus preform Selection <sup>1</sup> . . . . .	60
Figure 27 – Glass Preform UI with Ball preform Slection <sup>1</sup> . . . . .	61
Figure 28 – Glass Preform UI <sup>1</sup> . . . . .	62
Figure 29 – Glass Preform with Ball and Meniscus Plot <sup>1</sup> . . . . .	63
Figure 30 – Optical design UI <sup>1</sup> . . . . .	64
Figure 31 – Optical Design UI <sup>1</sup> . . . . .	65
Figure 32 – Sag Table and Preview Plot Feature <sup>1</sup> . . . . .	67
Figure 33 – Optical Design Modal UI <sup>1</sup> . . . . .	68

Figure 34 – Optical Design Modal UI <sup>1</sup> . . . . .	68
Figure 35 – Optical Design Modal UI <sup>1</sup> . . . . .	69
Figure 36 – Insert Modal with Flat Surface Type selected . . . . .	70
Figure 37 – Insert Modal Curvature <sup>1</sup> . . . . .	70
Figure 38 – Insert Modal Chamfer Table <sup>1</sup> . . . . .	71
Figure 39 – Insert Modal Curvature <sup>1</sup> . . . . .	71
Figure 40 – Molding Task Definition Form <sup>1</sup> . . . . .	72
Figure 41 – Optical Design Process cards . . . . .	73
Figure 42 – Optical Design side Bar . . . . .	74
Figure 43 – Process Side Bar . . . . .	74
Figure 44 – Molding Concept Modal <sup>1</sup> . . . . .	75
Figure 45 – Molding Concept Modal <sup>1</sup> . . . . .	75
Figure 46 – Image Mapper with Glass Preform option selected <sup>1</sup> . . . . .	76
Figure 47 – Image Mapper with Insert option selected <sup>1</sup> . . . . .	77
Figure 48 – Molding Task Plot <sup>1</sup> . . . . .	78

## LIST OF ACRONYM

**API** Application Programming Interface

**REST** Representational State Transfer

**FEM** Finite Element Method

**FEA** Finite Element Analysis

**GUI** Graphical User Interface

**IPT** Institute for Production Technology

**PGM** Precision Glass Molding

**UML** Unified Modeling Language

**PDEs** Partial Differential Equations

**CAD** Computer-Aided Design

**HTTP** Hypertext Transfer Protocol

**ODM** Object-Document Mapper

**JWT** JSON Web Token

**DOM** Document Object Model

**UI** User Interface

**HTML** HyperText Markup Language

**URL** Uniform Resource Locator

## CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>14</b>
1.1	MOTIVATION AND CONTEXT	14
1.2	SIMPGM PROJECT	15
<b>1.2.1</b>	<b>The Software</b>	<b>16</b>
1.3	THESIS OBJECTIVES	17
1.4	DOCUMENT STRUCTURE	17
<b>2</b>	<b>THEORICAL BACKGROUND</b>	<b>19</b>
2.1	PRECISION GLASS MOLDING	19
2.2	FINITE ELEMENTS SIMULATION	20
<b>2.2.1</b>	<b>Abaqus and Finite Element Analysis Software</b>	<b>20</b>
2.3	BACKEND DEVELOPMENT	22
<b>2.3.1</b>	<b>REST API</b>	<b>22</b>
<b>2.3.2</b>	<b>MongoDB</b>	<b>23</b>
<b>2.3.3</b>	<b>Object-Document Mapper (ODM)</b>	<b>23</b>
<b>2.3.4</b>	<b>JWT tokens</b>	<b>23</b>
2.4	FRONTEND DEVELOPMENT	24
<b>3</b>	<b>FRAMEWORKS</b>	<b>26</b>
3.1	API DIAGRAM	26
3.2	FLOW DIAGRAMS	26
3.3	USER CASE DIAGRAM	26
3.4	CONCEPTUAL JUSTIFICATIONS	26
<b>3.4.1</b>	<b>Alignment with Research Objectives</b>	<b>26</b>
<b>3.4.2</b>	<b>Flow Diagrams</b>	<b>26</b>
<b>4</b>	<b>SIMPGM SOFTWARE</b>	<b>28</b>
4.1	PROJECT REQUIREMENTS	28
4.2	FUNCTIONALITIES	28
<b>4.2.1</b>	<b>Authentication of Multi-tenant application</b>	<b>28</b>
<b>4.2.2</b>	<b>UI Development</b>	<b>30</b>
<b>4.2.3</b>	<b>Tool Creation</b>	<b>30</b>
<b>4.2.4</b>	<b>Tool Creation Options</b>	<b>30</b>
4.2.4.1	Create Tool from Scratch	31
4.2.4.2	Create Tool Based on Registered Tool	31
<b>4.2.5</b>	<b>Database Operations</b>	<b>31</b>
<b>4.2.6</b>	<b>Tools Preview Plot</b>	<b>31</b>
<b>4.2.7</b>	<b>Modal Features</b>	<b>32</b>
4.2.7.1	Surface Chamfer Table	32
4.2.7.1.1	Validation Process	33

4.2.7.2	Preview Surface Plot and Sag Table . . . . .	33
<b>4.2.8</b>	<b>Molding Task . . . . .</b>	<b>33</b>
4.2.8.1	Molding task plot . . . . .	34
<b>4.2.9</b>	<b>Error Handling . . . . .</b>	<b>34</b>
<b>5</b>	<b>IMPLEMENTATION . . . . .</b>	<b>35</b>
5.1	PROJECT STRUCTURE . . . . .	35
<b>5.1.1</b>	<b>Backend . . . . .</b>	<b>35</b>
5.1.1.1	Core /. . . . .	36
5.1.1.2	Helpers ./ . . . . .	37
5.1.1.3	Middleware ./ . . . . .	37
5.1.1.4	Models ./ . . . . .	37
5.1.1.5	Routers ./ . . . . .	37
5.1.1.6	Services . . . . .	37
<b>5.1.2</b>	<b>Frontend . . . . .</b>	<b>38</b>
<b>5.1.3</b>	<b>Components . . . . .</b>	<b>38</b>
5.1.3.1	Auth . . . . .	38
5.1.3.2	DatabaseGenerator . . . . .	39
5.1.3.3	SimulationGenerator . . . . .	39
5.1.3.4	SimulationVisualizer . . . . .	40
5.1.3.5	UI . . . . .	40
<b>5.1.4</b>	<b>Contexts . . . . .</b>	<b>40</b>
<b>5.1.5</b>	<b>API . . . . .</b>	<b>40</b>
5.1.5.1	Creating an Axios Instance: . . . . .	40
5.2	UI DEVELOPMENT . . . . .	41
<b>5.2.1</b>	<b>Reusable Components . . . . .</b>	<b>42</b>
5.2.1.1	<b>Example Implementation: . . . . .</b>	<b>43</b>
5.2.1.1.1	<i>Parent Component: Card . . . . .</i>	<i>43</i>
5.2.1.1.2	<i>Children Component: Definition form . . . . .</i>	<i>43</i>
5.2.1.1.3	<i>Usage in the Application: . . . . .</i>	<i>44</i>
5.2.1.1.4	<i>UI result . . . . .</i>	<i>44</i>
5.3	DATABASE GENERATOR IMPLEMENTATION . . . . .	45
<b>5.3.1</b>	<b>Tools Creation Process . . . . .</b>	<b>46</b>
<b>5.3.2</b>	<b>Getting User Data . . . . .</b>	<b>49</b>
<b>5.3.3</b>	<b>Generators from Group 1: Project, Coating, Machine, and Process Parameters . . . . .</b>	<b>49</b>
<b>5.3.4</b>	<b>Generators from Group 2: Sleeve, Holder, and Cooling Plate . . . . .</b>	<b>51</b>
5.3.4.1	PDF Visualizer Component . . . . .	51
5.3.4.2	Drop-down Data . . . . .	54
5.3.4.3	UI . . . . .	55

<b>5.3.5</b>	<b>Group 3: Glass Preform</b> . . . . .	<b>58</b>
5.3.5.1	Preview Plot Component . . . . .	58
5.3.5.2	Glass Preform UI . . . . .	60
<b>5.3.6</b>	<b>Group 4: Optical Design and Insert</b> . . . . .	<b>63</b>
5.3.6.1	Chamfer Table . . . . .	65
5.3.6.2	Sag Table and Preview Plot Features . . . . .	66
5.3.6.3	Optical Design Surface UI . . . . .	67
5.3.6.4	Insert UI . . . . .	69
5.4	MOLDING TASK IN SIMULATION MANAGER MODULE . . . . .	71
<b>5.4.1</b>	<b>Definition Form</b> . . . . .	<b>72</b>
<b>5.4.2</b>	<b>Molding Task Side Bar</b> . . . . .	<b>73</b>
<b>5.4.3</b>	<b>Molding Concept</b> . . . . .	<b>74</b>
<b>5.4.4</b>	<b>ImageMapper</b> . . . . .	<b>76</b>
5.4.4.1	Overview of react-img-mapper . . . . .	76
5.4.4.2	Image Mapper . . . . .	76
<b>5.4.5</b>	<b>Preview Molding Task Plot</b> . . . . .	<b>77</b>
<b>6</b>	<b>CONCLUSION</b> . . . . .	<b>79</b>
6.1	CONCLUSIVE SUMMARY . . . . .	79
6.2	FUTURE WORK . . . . .	79
	<b>BIBLIOGRAPHY</b> . . . . .	<b>81</b>
	<b>APPENDIX A – APPENDIX WITH CLASSIFIED DATA</b> . . . . .	<b>85</b>
A.1	CORE ./ - GET_CURRENT_USER FUNCTION . . . . .	85
A.2	HELPERS ./ GLASS_PREFORM_COORDS EXAMPLE . . . . .	86
A.3	MIDDLEWARE ./ - CORS.PY FILE . . . . .	86
A.4	MODELS ./ - PROJECT MODEL . . . . .	87
A.5	ROUTERS ./ - PROJECT ENDPOINTS . . . . .	88
A.6	SERVICES ./ - PROJECT SERVICES . . . . .	89
A.7	CONTEXTS - MODAL CONTEXT EXAMPLE . . . . .	91
A.8	GETTING USER DATA - DATA CAPTURE EXAMPLE . . . . .	92
A.9	PDF VISUALIZER CODE IMPLEMENTATION . . . . .	93
A.10	DROP-DOWN DATA IMPLEMENTATION . . . . .	95
A.11	PREVIEW PLOT IMPLEMENTATION . . . . .	96

## 1 INTRODUCTION

In recent years, the development of web-based applications has seen a significant surge, driven by the accessibility, scalability, and collaborative potential offered by online platforms. Concurrently, Finite Element Method (FEM) simulations have become a cornerstone in engineering, enabling the analysis and prediction of complex physical phenomena across various domains. The integration of these two realms — web technology and FEM simulations — presents a promising avenue for advancing computational engineering tools.

The objective of this bachelor thesis is to design and implement one of the three modules of a web-based application specifically for conducting FEM simulations. By leveraging the power of modern web technologies, including interactive user interfaces and advanced software architecture, this project aims to streamline the process of setting up and preparing data for future FEM simulations. The goal is to enhance the accessibility and usability of FEM tools for engineers and researchers, making the use of simulations more user-friendly.

### 1.1 MOTIVATION AND CONTEXT

The manufacturing industry, particularly in the field of glass processing, has witnessed a paradigm shift towards advanced computational techniques for process optimization and quality control. Glass molding processes, which allows the manufacturing of complex glass lenses with free-form surfaces as well as micro optics (MENZ, et al., 2006) [1]. This process necessitate a robust simulation tools to predict and mitigate shape deviations, ensuring precision and efficiency in production. The most robust simulation tools use FEM. Which has been widely applied in computation of structural mechanics, solid mechanics, fluid mechanics, and thermodynamics. It is an important method to solve partial differential equations (WANGLi-li , in Foundations of Stress Waves, 2007) [2]. At the forefront of computational engineering tools stands ABAQUS software, renowned for its prowess in Finite Element Analyses (FEA) and simulation capabilities across diverse engineering domains. However, the complexity of setting up and running simulations within ABAQUS, coupled with the technical expertise required, poses a barrier to entry for many engineers and researchers in the optics industry.

The motivation behind this bachelor thesis is driven by the need to bridge the gap between advanced simulation technologies and practical usability. This project aims to develop a web-based application for glass molding process simulations using ABAQUS. The software is designed to streamline the simulation workflow, providing intuitive interfaces for configuration, parameter input, simulation execution, results visualization, and store projects information.

## 1.2 SIMPGM PROJECT

This thesis is part of Fraunhofer Institute for Production Technology (IPT) in Aachen, North Rhine-Westphalia, Germany. The project is supported by the Fine Machining & Optics Department, which focuses on developing solutions in various fields related to the manufacturing and production of glass optics.

The project conducted involves the new version development of a web-based application designed for the smart production of optical components. This application aims to assist users in creating simulations easily and intuitively, as well as summarizing the simulation results. This software will empower engineers and researchers, regardless of their computational expertise, to harness the potential of ABAQUS for optimizing lens manufacture processes. In the diagram presented on the Figure 1 we can see the main functional requirements of the software.

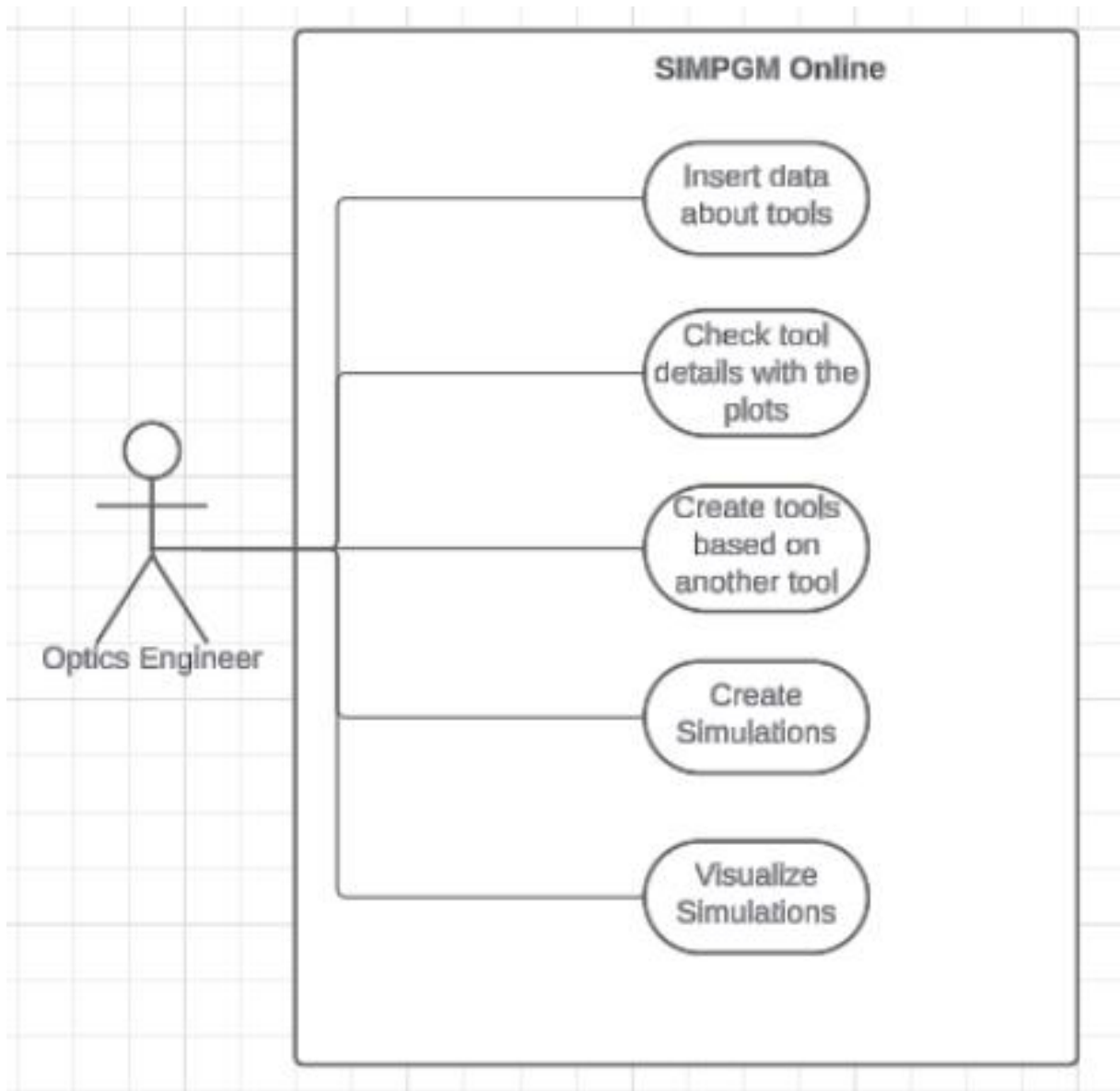


Figure 1 – Use Case Diagram with the main functional requirements of the software



### 1.2.1 The Software

The SIMPGM software has 3 main modules:

- Database Generator,
- Simulation Generator
- Simulation visualizer Its possible to see a visual representation of them in the Figure 2:

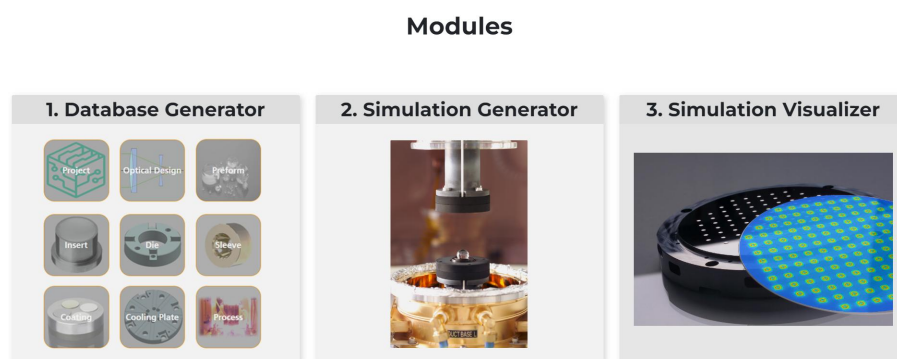


Figure 2 – SIMPGM Old Version Interface <sup>1</sup>.

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

**Database Generator:** The initial step for a user when using the software for the first time is to utilize the Database Generator module. Within this module, users generate all the necessary data essential for use in the other modules. Which includes creating the data for Project, Optical Design, Glass Preform, and tools such as Coating, Sleeve, Cooling Plate and Holder.

**Simulation Generator:** The next step for a user, after generating the initial data, is to utilize the Simulation Generator module. In this module, users can gather the data created in the Database Generator, send it to ABAQUS to run the simulations, and efficiently manage the entire simulation process. This includes organizing simulation tasks and monitoring progress by ABAQUS.

**Simulation visualizer:** In the Simulation Visualizer module, users can explore simulation results and details of the molding process. This module allows for an in-depth exploration of the simulation, providing visual insights and a comprehensive understanding of the molding dynamics and outcomes. In the Figure 3 its possible to see the Simulation Visualizer Interface.

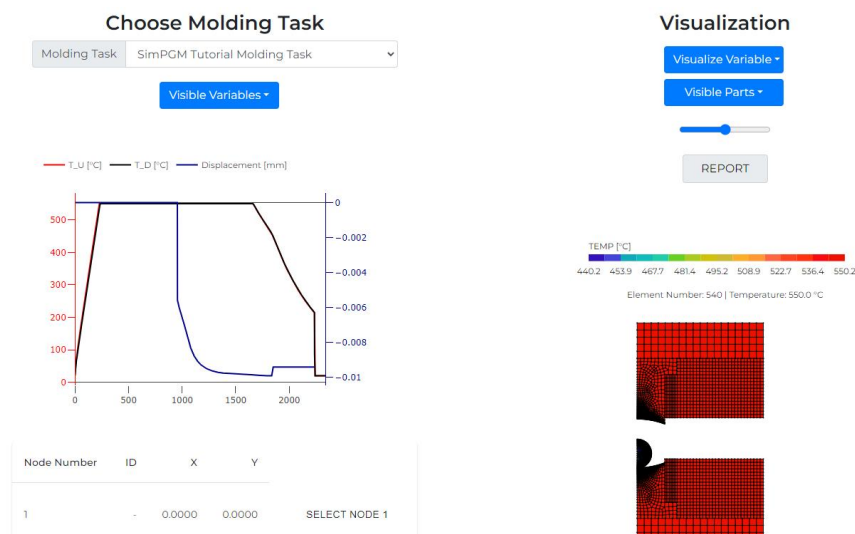


Figure 3 – Simulation Visualizer Module First Version Interface <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

### 1.3 THESIS OBJECTIVES

**Specific Objectives** To achieve the general objectives and work proposal proposal outlined in the introduction, it is essential to accomplish the specific objectives.

- Develop the Backend API endpoints for the Database Generator and Molding Task Feature
- Design and implement a multi-tenant architecture using MongoDB
- Develop the Frontend API Connection
- GUI Application and Responsiveness for the entire application

### 1.4 DOCUMENT STRUCTURE

**Introduction:** This chapter introduces the context and motivation behind the development of SIMPGM online. It outlines the objectives and the project modules.

**Theoretical Background:** Chapter 2 delves into the theoretical underpinnings essential for understanding the project. It covers fundamental concepts of FEM, web application development, and relevant technologies used in the implementation.

**Configuration Framework Concept:** Chapter 3 presents a comprehensive review of the concepts that guided the development. We delve into the theories, models, techniques, and methodologies that inform the problem-solving approach undertaken in this research

**SIMPGM software:** Chapter 4 It outlines the project requirements, assumptions, and the overall architecture of the configuration framework. Additionally, this chapter details the configuration process, including step-by-step procedures.

**Implementation:** Chapter 5 covers the practical implementation of the configuration framework using selected programming languages and frameworks. It discusses the backend API development, frontend GUI design, and preparing the data for FEM simulation.

**Conclusion and Future Work:** The final chapter summarizes the key findings, conclusions, and contributions of the project. It reflects on the accomplishments in developing the web-based FEM simulation application and outlines potential avenues for future enhancements, research, and collaborations within the domain of glass process molding and computational engineering.

## 2 THEORETICAL BACKGROUND

Chapter 2 provides a theoretical background essential for understanding the project's context and methodologies. The first section outlines Precision Glass Molding, emphasizing key techniques and methodologies. The second section explores Finite Elements Simulation and Web Deveoloment, summarizing fundamental concepts. Together, these sections lay the foundation for the subsequent discussions and analyses in the thesis, providing necessary context and understanding for the project's objectives and methodologies.

### 2.1 PRECISION GLASS MOLDING

Precision glass molding (PGM) stands as a thermal compression method utilized in crafting optical components from inorganic glasses. It's specifically ideal for cost-effective production of intricate, high-precision, mid- to high-volume optical elements such as single lenses, lens arrays, and lens on wafer scales. This process involves a sequence of five primary steps: Loading, Heating, Pressing, Cooling, and Unloading (ANH TUAN VU, et al.)[3].

Commencing with the loading of a pre-fabricated glass preform into the chamber, the preform undergoes heating by the mold, powered by infrared lamps. Temperature monitoring is facilitated by thermocouples within the dies, relaying data to the machine's PID controller. This ensures precise and swift attainment of the molding temperature, maintained within the temperature range between the transition temperature ( $T_g$ ) and the softening temperature.

Subsequently, the chamber maintains this temperature for a specific duration, ensuring uniform temperature distribution across the glass preform. The lower mold ascends, driven by a servo motor, applying the designated pressing force and molding time to shape the glass preform to the desired center thickness. Gradual cooling commences with regulated  $N_2$  gas flow, reducing the temperature below  $T_g$  while maintaining a holding force on the glass to minimize shrinkage. Upon force release, rapid cooling ensues, culminating in the unloading of the molded lens for inspection.

Accompanying this explanation is a graph detailing the process on Figure 4, illustrating temperature and force curves experienced throughout the molding procedure.

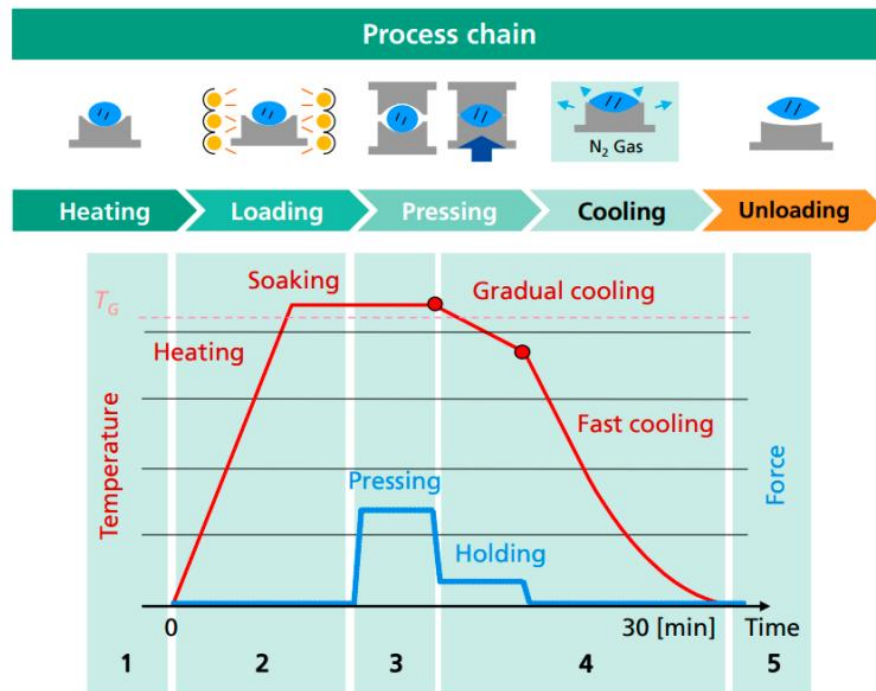


Figure 4 – Precision Glass Molding Process

## 2.2 FINITE ELEMENTS SIMULATION

FEM is a numerical technique used in engineering and scientific computations to solve partial differential equations (PDEs) and analyze complex systems. It has become a cornerstone in computational engineering due to its versatility and ability to handle problems with irregular geometries, material heterogeneity, and complex boundary conditions.

At its core, FEM relies on the principle of discretization, where a continuous physical domain is divided into smaller, finite elements. These elements are interconnected at discrete points called nodes, forming a mesh that approximates the original domain. By representing the system with finite elements, FEM transforms the continuous differential equations into a system of algebraic equations that can be solved using numerical methods.

### 2.2.1 Abaqus and Finite Element Analysis Software

FEA software is a sophisticated computational tool that engineers and scientists use to simulate and analyze complex physical phenomena. The fundamental principle behind FEA is the decomposition of a large, complex system into smaller, manageable pieces, known as finite elements. These elements can be in the form of triangles, quadrilaterals, tetrahedra, or other shapes, which together form a mesh that approximates the geometry of the physical system.

These are the main steps of FEM simulation:

- Model Setup
- Meshing
- Material Properties and Boundary Conditions
- Solving
- Post-Processing
- Validation

1. **Model Setup:** The first step in an FEA simulation is to create a digital model of the physical system. This involves defining the geometry, which can be done by importing from CAD software or building it within the FEA software itself.
2. **Meshing:** Once the geometry is defined, the model is divided into finite elements. The meshing process involves the creation of nodes and elements. The finer the mesh (more elements and nodes), the more accurate the simulation, but also the more computational resources are required.
3. **Material Properties and Boundary Conditions:** Assigning material properties to the elements which define how the material will react under certain conditions. Boundary conditions define how the model interacts with its environment, such as where there are supports or loads.
4. **Solving:** The software applies mathematical equations that govern physical behaviors (like the Navier-Stokes equations for fluid dynamics or the elasticity equations for structural analysis) to each element. These equations account for the material properties and boundary conditions. The solver uses numerical methods, such as the FEM, to approximate the solution of these equations.
5. **Post-Processing:** After the solution is obtained, results are available for analysis. Post-processing tools within the software allow users to visualize and interpret the results in the form of displacements, stresses, temperatures, fluid flows, and other physical quantities.
6. **Validation:** The simulated results are often validated with experimental data or analytical solutions to ensure the accuracy of the FEA model.

FEA software packages widely vary in their capabilities, with some focused on specific applications like structural analysis, thermal analysis, or fluid dynamics, while others are more general-purpose. Examples of FEA software include ANSYS, Abaqus, SOLIDWORKS Simulation, and COMSOL Multiphysics.

FEA software enables engineers to predict how products will react to real-world forces, vibration, heat, fluid flow, and other physical effects. This predictive capability can significantly reduce the time and costs associated with the prototyping and testing of new products. It also plays a critical role in optimizing designs for performance, safety, and compliance with regulatory requirements.

## 2.3 BACKEND DEVELOPMENT

The backend serves as the backbone of the application, handling data storage, server-side logic, API integration, and communication between the frontend. The following subsections describe the fundamental technologies and methodologies employed in the development of the backend system.

### 2.3.1 REST API

Representational State Transfer (REST) is an architectural style introduced by Roy Fielding in his doctoral dissertation that outlines a set of constraints for designing scalable web services. RESTful APIs that follow this architectural style enable stateless client-server communication, primarily using HTTP methods such as GET, POST, PUT, and DELETE.

In the context of our web application, REST APIs are instrumental in facilitating the interaction with the backend and frontend of the application. These APIs are designed to handle requests from the client side to perform operations on the server. Through the use of endpoints, the application provides a means for users to create, read, and update data pertaining to their simulations.

REST APIs are defined by six key architectural constraints: (1) Client-server: Separates clients and servers via a uniform interface. (2) Stateless: No client context is stored on the server; each request contains all necessary information. (3) Cacheable: Clients can cache responses to enhance performance. (4) Layered system: Clients can't tell if they're connected directly to the server or through intermediaries. (5) Code on demand: Servers can temporarily extend client functionality by transferring executable code. (6) Uniform interface: All resources are identified by URIs in requests, ensuring a consistent design (MARIO ANDRÉS PAREDES-VALVERDE, et al.)[4].

The creation of user simulation data is a critical feature of the application, allowing for a personalized and interactive experience. Users can input specific parameters for their simulations, which are then processed by Abaqus to generate the corresponding results. The API ensures that these data transactions are smooth and efficient, providing a bridge between the user interface and the computational power of the simulation software.

### 2.3.2 MongoDB

MongoDB is a NoSQL database that provides high performance, high availability, and easy scalability. It is an open-source document database that stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time (MongoDB Organization). This model allows for the storage of complex hierarchies and is particularly adept at handling large volumes of unstructured data.

In the context of our web application, MongoDB serves as the data storage solution for managing user simulation data, as well as other application-specific data. Its schema-less nature offers the flexibility needed to store diverse datasets generated from Abaqus simulations. MongoDB's powerful query language enables efficient data retrieval, which is crucial for providing real-time feedback to users and handling complex queries.

### 2.3.3 Object-Document Mapper (ODM)

An Object-Document Mapper (ODM) is a programming technique used to convert data between incompatible type systems in object-oriented programming languages. In the context of NoSQL databases, such as MongoDB, an ODM automates the conversion of JSON-like document data into objects that a programming language can use, and vice versa.

ODMs provide a high-level abstraction upon the database which encourages developers to think in terms of objects rather than database semantics. This abstraction simplifies the code and reduces boilerplate, thus enhancing development productivity. Additionally, ODMs often include additional layers of functionality such as validation, query building, business logic hooks, and more, which streamline application development processes.

### 2.3.4 JWT tokens

To enhance the application's security, a system of JSON Web Token JWT tokens was integrated, providing access and refresh keys. A JWT is a compact, representation to securely transferred information between two parties (BADR EDDINE SABIR, et al., 2019)[5]. This addition not only fortified the credential validation process but also significantly improved access management for customers. The server initiates the creation of a JWT, encoding and signing it with its secret key. The JWT token encompasses all encrypted user information, and its structure includes expiration information, thwarting any attempts by others to misuse the token for unauthorized requests.

To illustrate this process within the SIMPGM system, refer to the simplified diagram below on Figure 5:



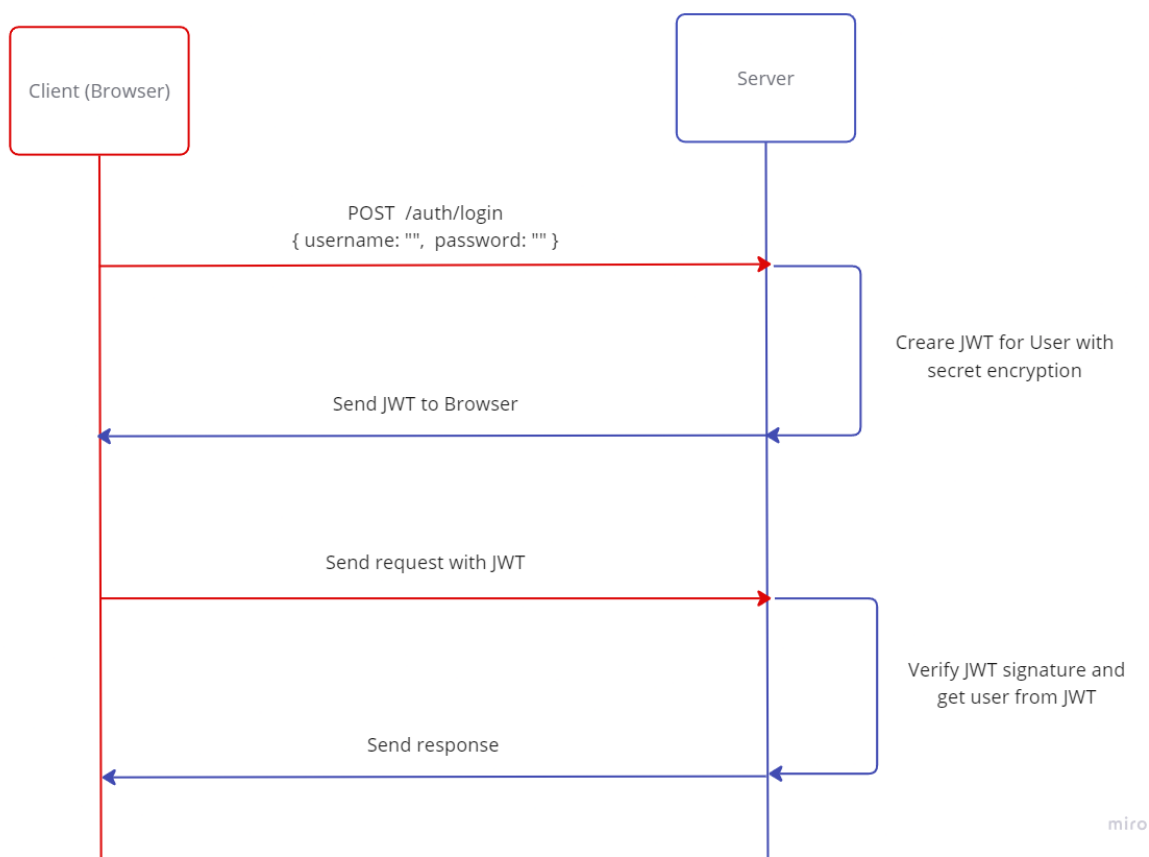


Figure 5 – JWT token server and User side

In the case of this project, we have two tokens: the user's access token, utilized only at the start of the session, and the refresh token, used when the user sends requests. Consequently, we required a distinct approach for storing the data schema for each token. These tokens are generated with information from TenantDB, transmitted to the user via cookies. With each request, the user sends their refresh token, which contains the expiry time of their session. The system then validates the information, authorizes the request, and sends the response to the user.

## 2.4 FRONTEND DEVELOPMENT

Interface development for the frontend of a web application involves designing and implementing the user-facing part of the application. The objective of the frontend interface development process is to offer the user an intuitive experience by creating interactive and responsive interfaces for a web application. This incorporates layout design, interactivity, and visual aesthetics of a web application to ensure that the end-user gets to perform the desired tasks efficiently and effectively.

Frontend development typically combines technologies such as HTML, CSS, and JavaScript to build an application's structure, style, and functionality. In this case, the front-end of the web application should represent complex data from the simulation in

an easy way for end-users, while it also offers a robust set of features within them: data insertion, visualization of results, and data management.

It means that in our web application, a dynamic responsive frontend will be built using ReactJS. Due to the component-based approach of React, it helps in breaking down the User Interface of the application into reusable components. This includes buttons, input fields, graphs for simulation data visualization, and much more, improving maintainability and scalability in the application.

One of the major features of React is its virtual Document Object Model, making the process of update and render of a user interface very efficient. This will be relevant for our application, as it demands a lot from the UI refreshes whenever new simulation data arrives or a user interacts with the application.

It has, in its ecosystem, libraries and tools such as Redux for state management and React Router for navigation, all of which can be integrated to add to the capabilities of the application. This will mean that these libraries provide clean, structured ways to handle the state and routing of the application and thus manage complexity as the application grows.

### 3 FRAMEWORKS

In this chapter, we explore the concepts that guided the formulation and development of the proposed solution. We explore the theories, models, techniques, and methodologies.

#### 3.1 API DIAGRAM

API diagrams are helpful to understand how the different pieces of your system communicate together and for data flow in your application. API diagrams help provide a visual understanding of endpoints, methods and the way data is passed between back-end for better clarity on how application is working underneath

#### 3.2 FLOW DIAGRAMS

Flow chart diagram uses a sequence of processes or activities that represents the visual representation in system-projects. It is made up of various shapes and lines that are used to represent the order in which tasks, decisions and results will occur. The flowchart diagram for our project will demonstrate workflows used in creating and executing molding tasks. It starts with data collection from the users by using interactive forms and modal dialogs.

#### 3.3 USER CASE DIAGRAM

User Case Diagram — Illustrates how users (actors) interact with a system. It shows the different use cases (actions) that can be done on a system by an actor. It served the following purposes for them: outlined system scope identified requirements communicated functionality to stakeholders

#### 3.4 CONCEPTUAL JUSTIFICATIONS

##### 3.4.1 Alignment with Research Objectives

These theories, models and techniques have been chosen because they are in line with the assumptions involved within our basic research goals. Each of these concepts was selected based on its potential to significantly contribute towards improving our knowledge and achieving the goals that we set in this research.

##### 3.4.2 Flow Diagrams

Flowchart Diagram is desirable to project because of its number one, it serves as a graphical guide so that the way you desire hence making things transparent either

via exhibiting all of the steps are involved in view that this created like step-by-step diagram. Even more importantly, it helps fellow teammates like to begin understanding what is all the work of project doing. Secondly, it is used as a lingua franca to make sure that other people understand the simple representation of complicated mechanisms.

## 4 SIMPGM SOFTWARE

This chapter outlines the development concept for the project functionalities. It begins by detailing the project requirements. Subsequently, the functionalities and the steps for their implementation are comprehensively described.

### 4.1 PROJECT REQUIREMENTS

This project aims to facilitate the creation and analysis of FEM simulations for the Precision Glass Molding process, and also work as tool to centralize information from the projects. The focus of this bachelor's thesis is on the database generation module, which is responsible for developing the tools that will be used to generate the simulations.

The database generation module is the most important module of the software. Through this module, we will create most of the necessary tools that will later be used in other modules, such as the simulation generator and the simulation visualizer.

To achieve this, eight functionalities were structured for the creation of the tools. The objectives of these functionalities are:

- Ensure the quality of the developed tools.
- Provide intuitively tool creating for the user.
- Ensure the security of each user's data.
- Allow the preview of the tools.
- Store information from the user's project's.

### 4.2 FUNCTIONALITIES

#### 4.2.1 Authentication of Multi-tenant application

This is not an exclusive functionality of the module but rather of the software as a whole. There are 3 different types of data storage strategy using the Multi-tenant approach. The first is *Separate Application, Separate Database*, where each user has their own software and their own database. The second is the *Separate Application, Separate Database* tenants use the same software and the same database. And the third one, which we are using is the *shared Application, Separate Database*. In the third approach, each each tenant has his own dedicated database, and shared the same application (GOZDE KARATAS BAYDOGMUS, et all) [6]). The current project use the third approach, because stores the user data separately, ensuring that one tenant cannot access another tenant's data. This isolation is crucial for our project because

maintaining privacy and security of our clients sensitive data especially when handling sensitive information.

To achieve this, it was necessary to implement an authentication and token management system for an application with multiple databases, which could potentially become very complex. To address this challenge, an approach was adopted to create a dedicated database for user authentication and access token storage, aptly named 'Tenant Database'.

The tenant database serves as a central hub for user authentication operations, covering processes such as login, registration, and the storage and validation of access tokens. Refresh tokens are stored in each respective user's database. Operations such as creating new users and other administrative tasks are limited to a single superuser known as Admin. This architecture simplifies the system and increases data security within the application. In the Figure 6 there is a visual explanation of a simplified version of the Multitenant Architecture.

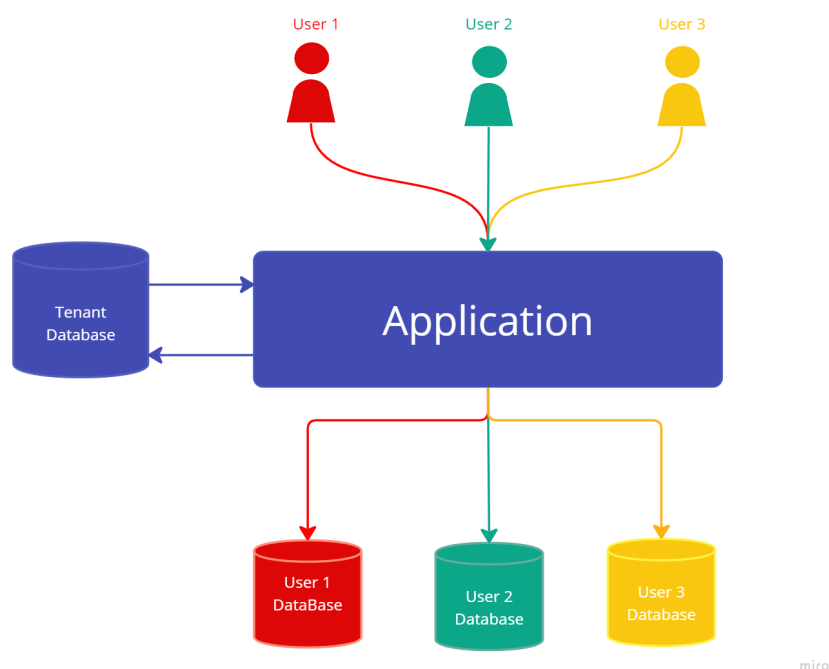


Figure 6 – Multitenant Architecture

The application assumes the responsibility of managing user database connections through an authentication service, utilizing a JWT token system, and providing access and refresh keys. The access keys also include information about which database can be accessed by that user. Thus, the server initiates the creation of a JWT, encoding and signing it with its secret key. The JWT token encompasses encrypted user information and includes expiration details, preventing any third-party attempts to use the token for unauthorized requests.

This way, client access management becomes more secure, and it also benefits from the Fraunhofer Institute's server, which has a robust security layer and a firewall.

### 4.2.2 UI Development

One of the aspects regarding the development of our web application appears to be the user interface UI design. Our UI development process is centered around enhancing the user experience through intuitive design principles.

Every element of our UI is meticulously crafted to ensure ease of use and clarity. Navigation within the application is designed to be intuitive and straightforward. We prioritize clear menu structures and logical pathways to help users find their way around the application effortlessly. The layout of each screen is considered to optimize usability.

Every UI component presented in the interface was developed by the author, using simple tools such as HTML, CSS, and JavaScript. These UI elements are designed to be intuitive and responsive. Buttons, links, and other interactive components provide clear feedback to users, helping them understand the actions they can take and the outcomes they can expect. Whether submitting a form, selecting options from a dropdown menu, or interacting with multimedia content, users are provided with a rich experience.

### 4.2.3 Tool Creation

The Database Generator Tools functionality includes the creation of API endpoints to enable the creation and retrieval of data. These endpoints serve as the interface between the frontend and the database, allowing users to interact with the system programmatically. The 'Create' endpoint enables users to input new data into the system, while the 'Get' endpoint retrieves data from the database based on specified criteria.

In addition to API endpoints, this functionality ensures seamless integration with the frontend interface for efficient data retrieval and insertion. Frontend components are developed to interact with the API endpoints, enabling users to access and manipulate data through a user-friendly interface. Users can access existing data and input new data directly through the frontend. These changes are reflected in the database in real time.

### 4.2.4 Tool Creation Options

In addition to enabling data creation and retrieval, the Database Generator Tools functionality offers users the flexibility to create tools from scratch or based on existing tools registered within the system. This feature enhances user convenience and efficiency by providing multiple options for tool creation tailored to individual user preferences and project requirements.

#### 4.2.4.1 Create Tool from Scratch

Users have the option to create a tool from scratch, allowing them to define the tool's specifications, parameters, and functionalities according to their specific needs. This approach provides maximum flexibility and customization, empowering users to design tools tailored to unique data generation and management requirements.

#### 4.2.4.2 Create Tool Based on Registered Tool

Alternatively, users can choose to create a tool based on a tool already registered within the system. This feature streamlines the tool creation process by leveraging existing templates or configurations, saving time and effort in tool development. Users can select a registered tool as a starting point and modify it as needed to suit their particular use case, accelerating the tool development process while maintaining flexibility and customization options.

By offering both options for tool creation, the Database Generator Tools functionality enhances user productivity and efficiency, allowing for seamless integration of new tools into the system while accommodating diverse user preferences and project requirements.

### 4.2.5 Database Operations

Efficiency in database operations is paramount for the smooth functioning and performance of any web application. Building upon the insights garnered from the internship report, our project places a strong emphasis on optimizing database interactions.

The strategic definition of API endpoints is pivotal in ensuring efficient database operations. With careful consideration, we design these endpoints to cater to specific functionalities and data retrieval needs. Our approach involves structuring GET endpoints to retrieve data with precision, minimizing unnecessary data fetching and processing. Similarly, the POST endpoints are meticulously crafted to handle data creation and updates swiftly and securely, mitigating potential bottlenecks and optimizing performance.

### 4.2.6 Tools Preview Plot

This plot functionality provides a very important method whereby users can actually view and analyze details of the tool they want to add into the system. This feature offers users the ability to observe the tool's parameters and characteristics based on the inputs provided, helping them ensure the accuracy and correctness of their data before adding it to the database.

An important functionality is the capability to visually represent complex structures and configurations of tools. Users can generate detailed plots that showcase



various aspects of the tool, including geometric dimensions, boundary conditions, and simulation results. Having these details visually represented can give users better insights into the behavior and performance that a tool should have.

Additionally, the plot functionality aids users in validating their inputs by providing visual feedback on the correctness and coherence of the provided data. Users can quickly identify any discrepancies or inconsistencies in their inputs by examining the plotted results, allowing them to rectify errors and refine their parameters before finalizing the tool creation process. By doing so, this validation step contributes to the security and ration of data within a system by avoiding errors or inaccuracies that crop up in further analyses and simulations.

The plot functionality also has an intuitive UI for interacting with plotted data. The user can zoom in the plots and check various aspects of the tool that helps to comprehend a good view about its features, and behavior.

#### 4.2.7 Modal Features

In order to allow the users to work with complex tools such as Optical Design and Insert. To address this requirement, we've developed exclusive features specifically to these tools, accessible through dedicated pages known as modals.

Modals serve as specialized environments within the application, providing users with a focused and streamlined interface for interacting with specialized features. These pages are designed to meet the specific requirements of each tool, offering users a comprehensive set of functionalities to define parameters and configurations.

The key feature of modals is the ability to input parameters for the surface design of specific lenses or tools. Users can interact with intuitive form fields and controls to specify various aspects of the surface geometry, such as curvature, thickness, refractive index, and optical properties. This enables users to customize the design of complex surfaces with precision and accuracy.

##### 4.2.7.1 Surface Chamfer Table

The chamfer table functionality within our web application provides users with a convenient way to define chamfer parameters for surfaces. This feature is particularly useful for users working with geometric designs or machining processes where chamfers are commonly applied.

Upon accessing the chamfer table, users are presented with two options: to add a slope or a rounded surface chamfer. This selection determines the type of parameters that will be required for defining the chamfer.

- If the user selects a slope, they are prompted to input parameters for the length of the slope and the angle of inclination.

- Alternatively, if the user selects a rounded surface chamfer, they only need to specify the radius of the rounded edge.

#### 4.2.7.1.1 Validation Process

The chamfer table includes a robust validation process to ensure that the defined chamfer parameters adhere to design constraints and best practices.

#### 4.2.7.2 Preview Surface Plot and Sag Table

This feature includes a crucial function aimed at improving user understanding and ensuring the accuracy of interface designs. This integrated functionality provides users with a holistic view of their surface designs, both visually and numerically, facilitating informed decision-making throughout the design process.

The inclusion of a surface plot visualization serves as a cornerstone for users to grasp the intricate details of their surface designs. By rendering the surface in a visual format, users gain insights into its shape, curvature, and overall appearance. This visual feedback enables users to intuitively assess the correctness of their inputs and parameters, empowering them to make informed design choices.

Moreover, the surface plot visualization serves as a powerful tool for validating user inputs and parameters. Users can scrutinize the plotted surface to identify any irregularities or discrepancies that may indicate errors in the design. By comparing the visual representation with their intended design, users can swiftly pinpoint areas requiring adjustment and ensure alignment with their specifications.

Complementing the visual feedback, users are provided with a comprehensive table containing numerical values corresponding to the plotted surface. This table offers detailed insights into the surface coordinates. By cross-referencing the numerical data with the visual plot, users can verify the accuracy of their designs and make necessary adjustments as needed.

The integration of surface plot visualization and parameter validation facilitates an iterative design approach, empowering users to refine their designs effectively. Users can iteratively adjust input parameters, observe the resulting changes in the surface plot, and validate modifications through numerical analysis. This iterative workflow enables users to fine-tune their designs incrementally until the desired surface geometry is achieved, enhancing design precision and efficiency.

### 4.2.8 Molding Task

The Molding Task feature is responsible for gathering the data and managing the workflow of creating a SIMPGM molding task. This feature has been made more

efficient by interfacing with different modules, including the database generator, to collect necessary information from other essential sources.

#### 4.2.8.1 Molding task plot

The Molding Task Plot feature is essential for visualizing the future simulation data effectively. It provides users with clear plots representing different aspects of their molding tasks. With interactive elements, users can explore the plots easily by zooming in or panning to focus on specific details. This feature helps users analyze their simulations and make informed decisions without unnecessary complexity.

### 4.2.9 Error Handling

Error handling is a critical aspect of any web application, ensuring that users receive appropriate feedback and guidance when unexpected situations arise. The error handling functionality in our web application is designed to provide users with clear and actionable messages to help them understand and resolve issues effectively.

The web app implements robust error handling mechanisms across various functionalities. Whether it's plot visualization, modal dialogs, tool creation, or user input validation, the system promptly detects and communicates errors to users. By providing informative error messages and proactive guidance, users can troubleshoot and resolve issues efficiently, ensuring a seamless and user-friendly experience.

In cases where errors prevent the application from functioning as expected, our web application displays user-friendly error pages to inform users of the issue and guide them towards resolution. These error pages are designed to be visually appealing and easy to understand, helping users to stay engaged and informed even in challenging situations.

## 5 IMPLEMENTATION

This chapter describes the project implementation. It begins by detailing the project structure, subsequently, each functionality implementation from the backend to the frontend.

### 5.1 PROJECT STRUCTURE

The project code is divided into two main structures: backend, and frontend to improve code organization and readability. Which one is going to be described in the sections below.

#### 5.1.1 Backend

This organizational structure was made it easier to centralize operations inside dedicated folders in a way that ensures the application is clean and supports reusable code. Each folder is assigned to specific functionality, for instance, database operations or definition of API endpoints, the codebase becomes much more efficient and easier to manage. This is important for maintainability as well as collaboration; the developer knows what each component deals with. In the package diagram in Figure 7 we can also understand the packages' relationship.

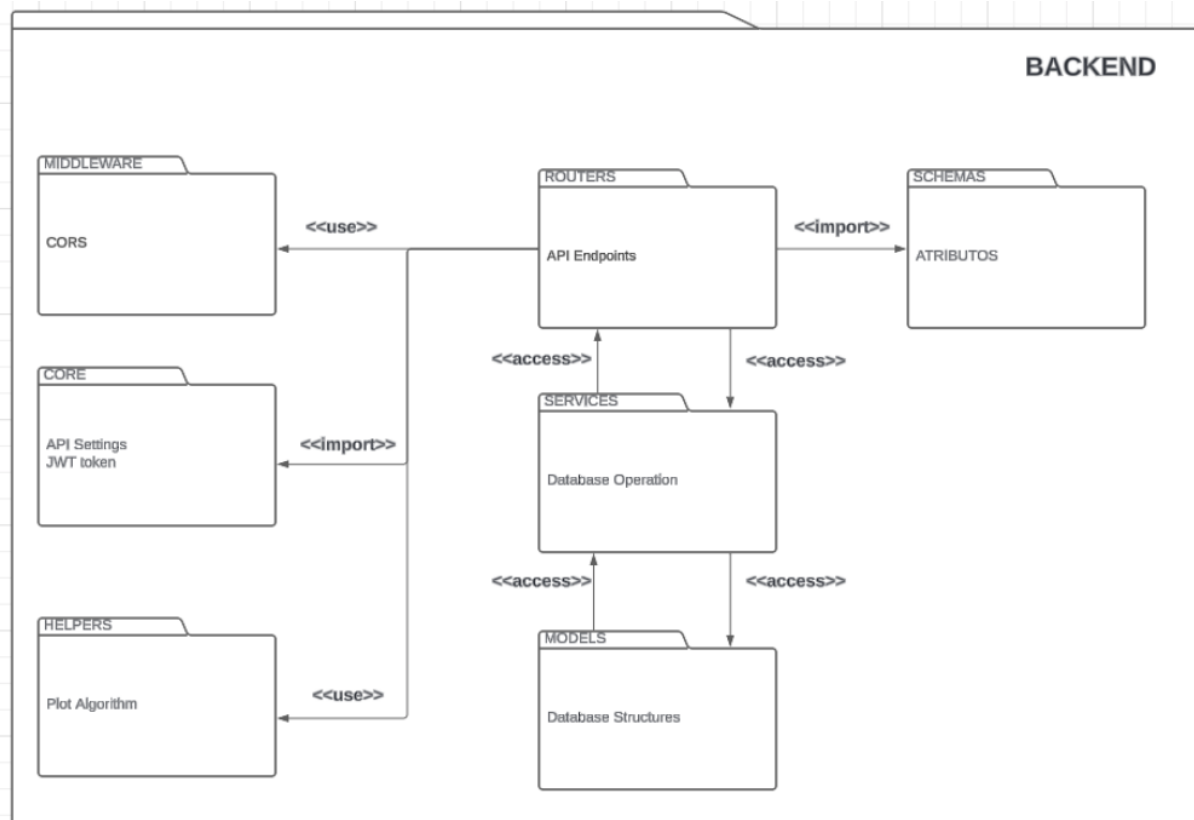
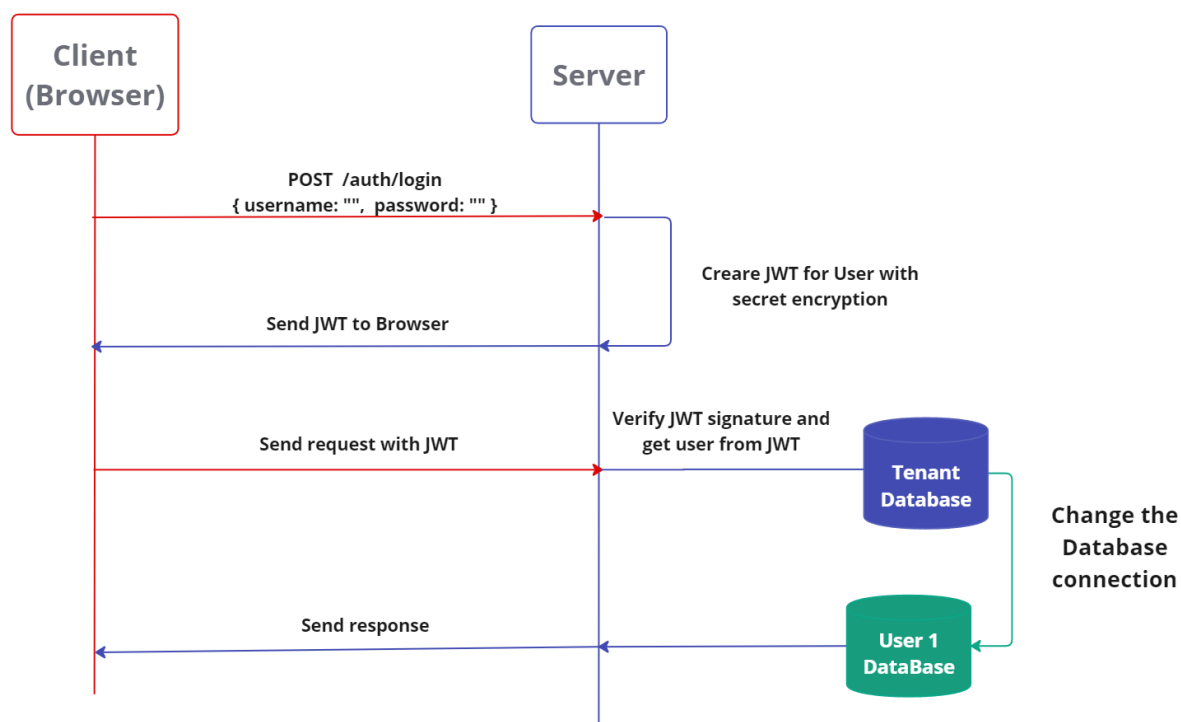


Figure 7 – Package Diagram Backend Structure

## 5.1.1.1 Core /.

In the **Core** folder, the **config.py** file stores essential information such as the database connection string, API string, project name, JKT token information, encryption algorithm, access token, and refresh token expiration details. Additionally, this directory contains the **JWTAuth.py** and **security.py** files. The former implements the logic for JWT token handling, while the latter manages credential-related operations.

The most important **Core** functionality is **get\_current\_user**, which is responsible for extracting data from the token, validating the JWT token, and ensuring that the user corresponds to the correct tenant database. This function performs critical security checks and manages database connections based on the user's identity. More details of the code can be explored in the appendix A.1. In the figure 8 we can also understand the validation logic:



miro

Figure 8 – JWT validation logic

#### 5.1.1.2 Helpers ./

Within this directory are Python functions responsible for generating coordinates for the tool, surface, and molding task plots. These files were developed by the author, who adapted the new project's data structure to the plotting algorithms developed by *Cheng Jiang, M.Sc. M.Eng*, the project supervisor. It's possible to see an example of the algorithms usage in the appendix A.2

#### 5.1.1.3 Middleware ./

The **Middleware** directory, the software facilitates communication or connectivity between applications or components in a distributed network. In this project, communication is enabled exclusively via the HTTP method to the frontend application, as declared in the ***CORS.py*** file. This file specifies the IP address, headers, and methods permitted to access backend information. More details about the middleware code implementation can be seen in the appendix A.3.

#### 5.1.1.4 Models ./

Within this directory, the database models are declared. These models serve as the intermediary between the Python application and the MongoDB database. Developed using the Beanie object-document mapper, each MongoDB document type has its own file, totaling 22 files. Within these files, a class is declared for each document type, specifying the fields and their types. Additionally, two class methods are included: one to calculate the lotID, an index for the documents, and another to calculate the "nr", representing the document number, along with a field to track tool versions. The details of the lotID calculation, nr calculation, and daa structure are in the Appendix with classified data, section A.4.

#### 5.1.1.5 Routers ./

This directory stores the endpoint information for the project. It declares the endpoint routes, specifying their URLs, HTTP methods, HTTP statuses, HTTP exceptions, and schema selections. The code below provides the endpoints for project in the Database Generator. More details about the code implementation can be seen in the appendix A.5

#### 5.1.1.6 Services

The services folder in the backend architecture is dedicated to handling database operations, ensuring a clear separation of concerns within the application. This folder contains service modules that are responsible for executing a variety of database tasks,

such as creating new records, retrieving specific data, and returning lists of registered elements. The details of the implementation can be seen in the appendix A.6

### 5.1.2 Frontend

The frontend structure holds together a host of components that are important for developing the user interface. These include UI configuration files, which are very important in configuring the different parts of the user interface, and a repository of UI components, where reusable elements are stored to be plugged in without much hassle in different parts of the application. This structure further includes style files that basically define how the UI components are going to look, hence ensuring both uniformity and aesthetics throughout the application. Lastly, the structure of the frontend includes modules to connect with APIs, thereby allowing easy and smooth communication between the frontend and backend systems.

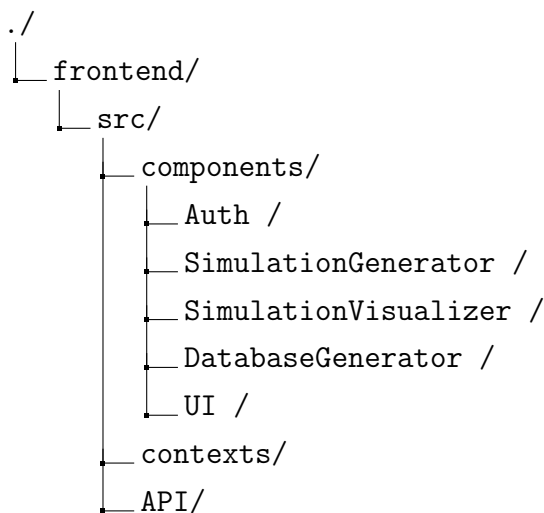


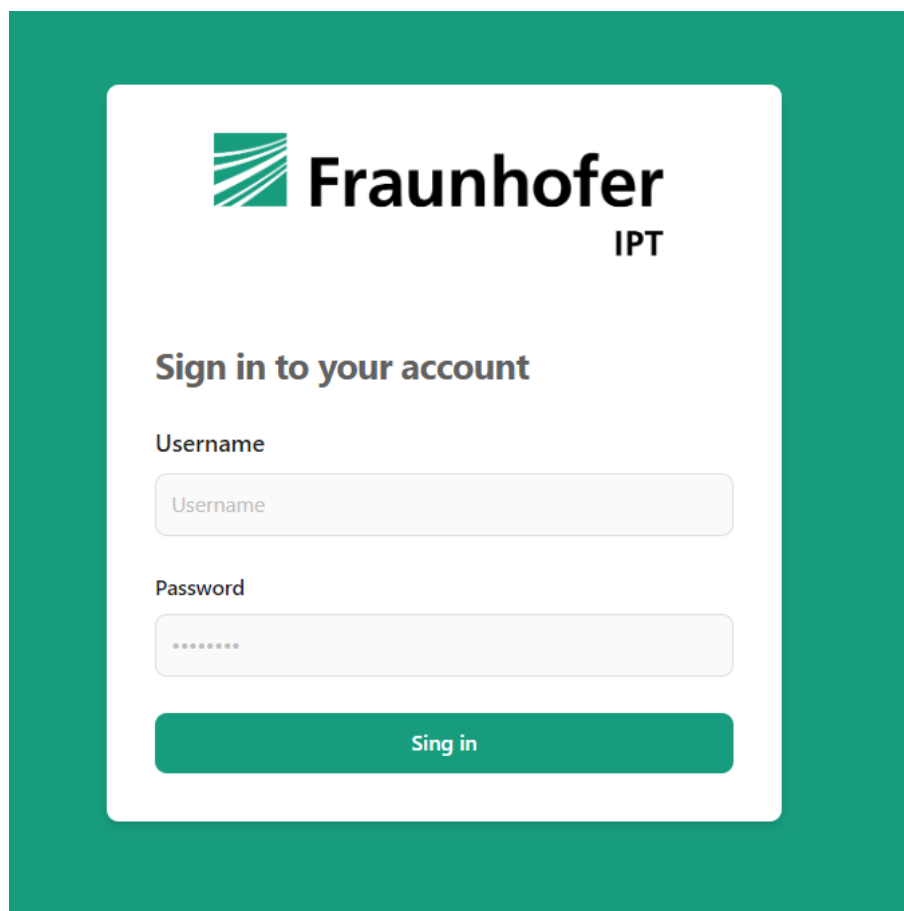
Figure 9 – Frontend Directory Structure

### 5.1.3 Components

The components folder in the frontend project is a crucial part of the application architecture. It is designed to organize and streamline the development of the user interface by modularizing various parts of the application. This folder is divided into several subfolders, each dedicated to a specific functional area of the application.

#### 5.1.3.1 Auth

The Auth folder in the React project contains components related to user authentication. This includes login forms, password reset interfaces, and any other components necessary for managing user access and authentication flows. By encapsulating these elements within a dedicated folder, the application ensures a secure and cohesive

Figure 10 – Login Form <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

user authentication experience. In the Figure 10 there is a example of the login form component.

#### 5.1.3.2 DatabaseGenerator

The **DatabaseGenerator** folder gather components responsible for creating and managing the database configuration and tool generation processes. These components provide user interfaces for defining database schemas, inputting required parameters, and setting up the necessary tools for simulation data management.

#### 5.1.3.3 SimulationGenerator

The **SimulationGenerator** folder includes components dedicated to setting up and running simulations. Users can input the necessary parameters, select tools and configurations, and initiate simulations through these components.



#### 5.1.3.4 SimulationVisualizer

The **SimulationVisualizer** folder contains components for visualizing simulation results. This includes graphing tools, plotting interfaces, and other visual aids that help users interpret and analyze the results of their simulations.

#### 5.1.3.5 UI

The **UI** folder encompasses general user interface components that are used across different parts of the application. This includes buttons, forms, modals, navigation bars, and other reusable UI elements. By centralizing these common components, the application promotes reusability and consistency in the user interface design.

### 5.1.4 Contexts

The **Contexts** folder in the frontend project is designed to manage the global state and shared logic of the application through the use of React Context API or other state management solutions. This folder includes various context providers that encapsulate state and behavior related to different parts of the application. More details of the Modal Context and useContext are in the A.7 inside the Classified Appendix.

### 5.1.5 API

The APIs folder in the frontend project is dedicated to managing the communication between the frontend application and the backend services. It encapsulates all the API calls, ensuring that data fetching and interactions with backend endpoints are organized, reusable, and maintainable.

The code snippet demonstrates the usage of Axios, a popular HTTP client for making requests from a web application to a server. In this case, Axios is used to facilitate communication between the frontend React application and a FastAPI backend server.

#### 5.1.5.1 Creating an Axios Instance:

An Axios instance is created using the `axios.create` method. The `baseUrl` option is set to the FastAPI server's address. This instance will use the specified base URL for all requests, simplifying the process of making API calls. The created Axios instance is exported as the default export. This allows other parts of the application to import and use this configured instance for making HTTP requests to the FastAPI server.

```
1 import axios from 'axios'
2
3 const fastAPIServer = "http://127.0.0.1:8000"
```

```
4
5 export const axiosInstance = axios.create({ baseURL: fastAPIServer })
6
7 export default axiosInstance;
```

Listing 5.1 – Using Axios configuration

By using this Axios instance, the application can easily perform various HTTP operations such as GET, and POST, to interact with the backend API. The configured `baseURL` ensures that all requests are directed to the correct server endpoint, promoting code reuse and consistency in API calls.

```
1 import axiosInstance from "../APISettings"
2 // POST endpoint
3 export const createProject = (data) => {
4   const endpointURL = "/project/create-project"
5   const new_project = {
6     // Data Structure Classified
7   }
8
9   return axiosInstance.post(
10     endpointURL,
11     new_project
12   )
13 }
14
15 // GET endpoint
16 export const getProjects = async () => {
17   const endpointURL = "/project/get-projects"
18   return axiosInstance.get(
19     endpointURL,
20   )
21 }
```

Listing 5.2 – Project API endpoints

## 5.2 UI DEVELOPMENT

Our objective is to make the FEM simulations setup and creation process more intuitive. In order to achieve it, the UI is designed to be user-friendly, and responsive making the application accessible across multiple devices and screen sizes. During

- **Interactive Feedback:** The design uses interactive visual elements like loaders, success messages or error alerts to provide immediate feedback to the user. This prevents users from remaining in the dark about their actions and potential errors.
- **Error Handling:** Comprehensive error handling mechanisms to control any possible threats in several functions. Errors about plot generation, modal dialogs,

tool creation and input validation are reported quickly to the user. Clear error messages and hints help to guide users in correcting these issues.

- **Intuitive Navigation:** The navigation structure is designed for Users to locate and use the different points in the application conveniently. This overall usability is further enhanced by clear labels, logical grouping and an intuitive navbar component.
- **Responsive Settings:** Users can access the application in different devices, the screen and design adapt to the user preferences and screen demands.

In the SIMPGM project, each user interface component was developed by the author. All components, and every piece of UI was developed by the author. Without using any external library, craft the entire fronted using only HTML, Tailwind CSS, and React. The UI is compose by Alerts, Buttons, Cards, Errors, Forms, Inputs, Modals, NavBar and SideBar. All elements will be displayed in the further examples;

### 5.2.1 Reusable Components

A very powerful feature from the React.JS framework is the component component-based architecture. This strategy allows developers to build the user interface by breaking it down into smaller, reusable components. Each component is essentially a self-contained piece of the UI that can be used multiple times throughout the application. In this pattern, components are designed to be compo-sable, meaning they can accept and render other components or content passed to them as "children." This allows developers to create highly reusable and flexible components that can adapt to different contexts and requirements.

According to MOCHAMMAD FARIZ SYAH LAZUARDY AND DYAH ANGGRAINI (2022)[7] React.js supports two types of components:

- **Class components:** were traditionally used to define stateful components in React and have a lifecycle management mechanism.
- **Functional components** are a more modern approach, especially with the introduction of Hooks (e.g., `useState`, `useEffect`), which allow you to manage state and lifecycle features without using class-based components.

In this the SIMPGM project we only used function components, due the more modern approach, community support, documentation recommendation, and comparability with different ReactJs and JavaScript modules.

An important aspect of reusable components is the children strategy. In essence, the children strategy involves parent component, a higher-order component that defines a structure or layout. And a children components, a nested components or content

passed from the parent, which the parent can render within its predefined structure. This aspect brought several advantages during the project development:

- **Modularity:** By breaking down the user interface into smaller, reusable components, the children strategy promotes a modular architecture. This makes the codebase easier to manage, understand, and maintain.
- **Reusability:** Components designed can be reused across different parts of the application, reducing redundancy and promoting consistency.
- **Flexibility:** The ability to pass dynamic content as allows a flexible and adaptable UI.
- **Separation of Concerns:** Helps in maintaining a clear separation between the layout and content, providing a cleaner and more organized code.

This approach led the SIMPGM application into various modules, each designed to handle specific functionalities such as user data input, simulation setup, and result visualization. By implementing it we achieve a flexible and maintainable UI and frontend code base. In the Figure 11 it's possible to see the strategy result in the UI.

### 5.2.1.1 Example Implementation:

#### 5.2.1.1.1 Parent Component: Card

```
1
2
3 export const Card = (props) => {
4   return(
5     <div className="lg:w-screen md:w-screen lg:mx-auto w-12/12 bg-
6       white rounded-lg shadow-xl md:mt-0 sm:max-w-md xl:p-0" >
7       {props.children}
8     </div>
9   )}
export default Card;
```

Listing 5.3 – React Parent Component

#### 5.2.1.1.2 Children Component: Definition form

```
1 import Card from "../..//Card/Card";
2 export const DefinitionForm = (props) => {
3   return (
4     <Card>
5       <div className=" flex flex-col item-center lg:p-12 p-16">
```

```
6         <h5 className="text-3xl font-medium text-center
7           text-gray-900">Definition</h5>
8         <div className="space-y-4 md:space-y-6 text-left" >
9           {props.children}
10        </div>
11      </div>
12    </Card>
13  )}
14 export default DefinitionForm;
```

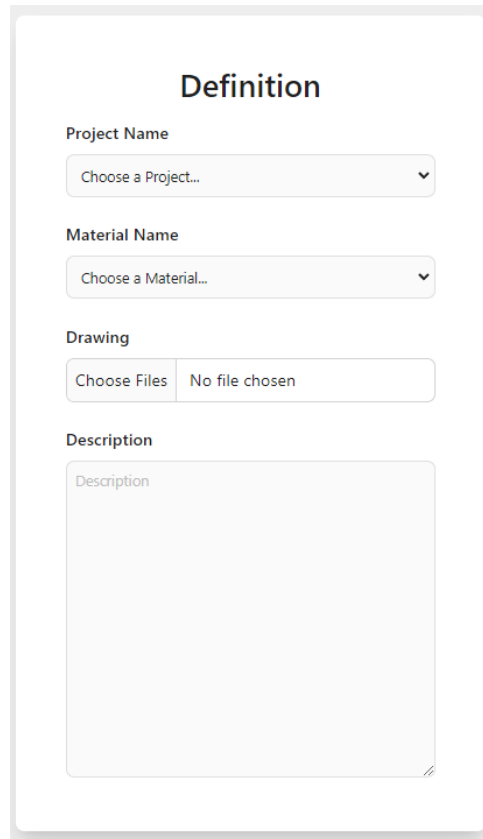
Listing 5.4 – React Chield Component

### 5.2.1.1.3 Usage in the Application:

```
1 <DefinitionForm>
2   <FormDropDownInput
3     label={"Project Name"}
4     targetName={"Project"}
5     defaultValue={inputValuesMapper["project_name"]} &&
inputValuesMapper["project_name"]}
6
7   />
8   <FormDropDownInput
9     label={"Material Name"}
10    targetName={"Material"}
11  />
12  <FormFileInput
13  />
14  <FormTextAreaInput
15  />
16
17 </DefinitionForm>
```

Listing 5.5 – Real Usage in the Application

### 5.2.1.1.4 UI result



The image shows a web form titled "Definition". It has four main sections:

- Project Name:** A dropdown menu with the text "Choose a Project..." and a downward arrow.
- Material Name:** A dropdown menu with the text "Choose a Material..." and a downward arrow.
- Drawing:** A button labeled "Choose Files" followed by a text box containing "No file chosen".
- Description:** A large text area with a light gray background and the placeholder text "Description".

Figure 11 – Definition Form Implementation Result <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

### 5.3 DATABASE GENERATOR IMPLEMENTATION

The database generator is the first module that the users encounter. This module enables the user to create most of the data generated during the simulation. Each page in this module concerns the user with data generation for a particular part or process, thus called the generators. There are about ten generators in total, including: Project, Optical Design, Glass Preform, Coating, Insert, Sleeve, Holder, Cooling Plate, Machine, and Process Parameter. It's worth mentioning that materials and flange data are not generated in this module, since they are directly inserted into the user's database.

Each generator provides its unique data to the users. However, it's possible to classify the generators according to their complexity. We could simplify them into the following classification:

- **Group 1: Project, Coating, Machine, and Process Parameters**
- **Group 2: Sleeve, Holder, and Cooling Plate**
- **Group 3: Glass Preform**

- **Group 4: Optical Design and Insert**

### 5.3.1 Tools Creation Process

Before delving into the tool groups, it's essential to understand the tool creation process. There are two primary methods for creating a tool: Create from Scratch or Create based on an existing tool. Although similar, these methods require different approaches. In the Figure 12 and Figure 13 it shows the Create Options Pop up for the user.

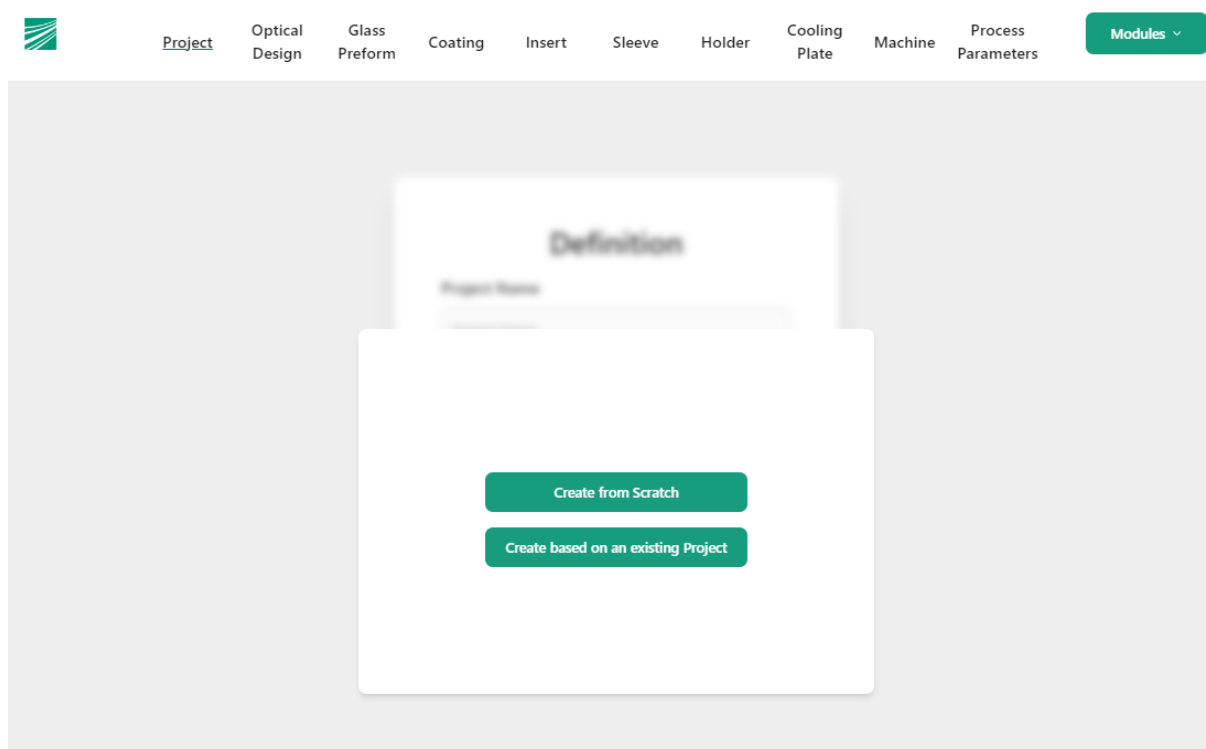


Figure 12 – UI screenshot with two creation options for the user <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

**Create from Scratch:** When opting to create from scratch, the user will have only to insert the data on the inputs.

**Create based on a Previous Tool:** However, if the user prefers to create a tool based on an existing one in the database, the inputs values will be change based in the select tool.

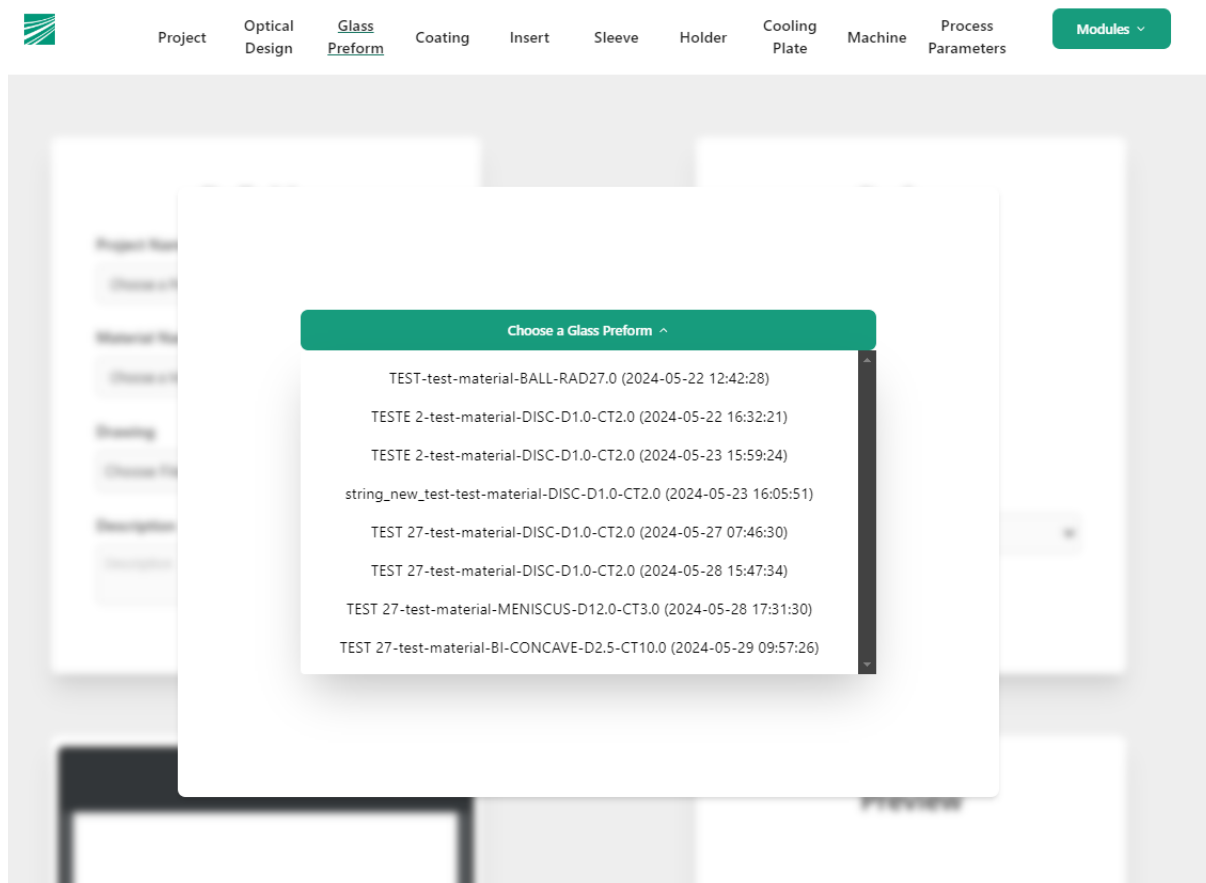


Figure 13 – UI screenshot with Project and Machine Generators

In the example bellowed we can see the *setValuesFromBasedTool* example, which sets the default value for the inputs from the database . This function will be called everytime the Create based on a Previous Tool Modal is selected. In the Figure 14 we can have a better understanding of the process.

```

1 const setValuesFromBasedTool = async (bsonData) => {
2   setEditMode(true)
3   setInputValuesMapper(bsonData) // Sets the inputValueMapper
   useState responsbale for defining the default value for a user
4 }

```

Listing 5.6 – Real Usage in the Appllication



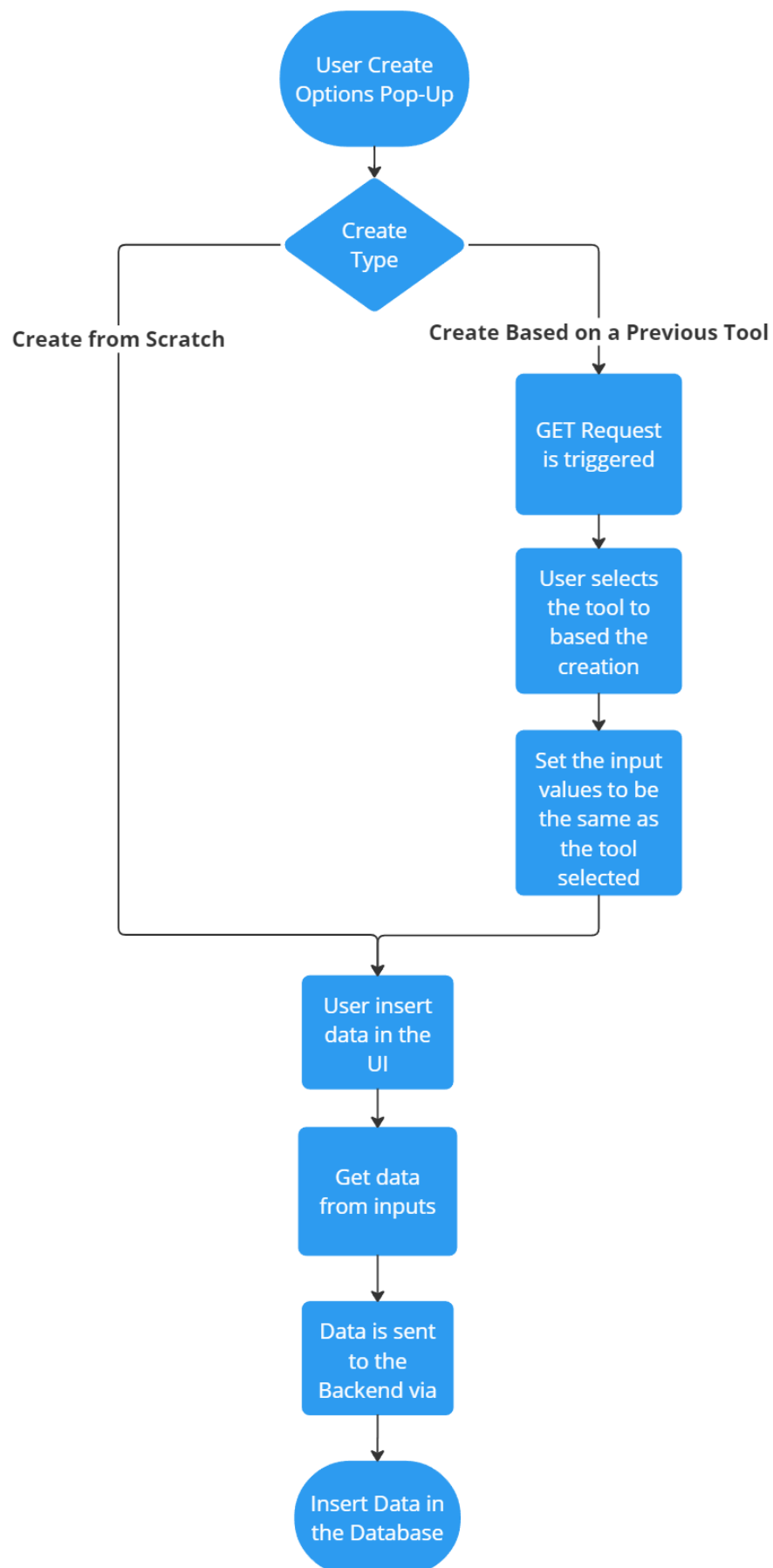


Figure 14 – Create Option flow chart

### 5.3.2 Getting User Data

The data from this and all generators is captured with the help of the **useForm** package from React. This package simplifies the process of capturing user data by providing methods such as **register** and **handleSubmit**.

In the code below, we provide an example using the Project Generator. We import the **useForm** package and extract the **register** and **handleSubmit** methods using the **useForm()** hook. The **register** method is configured to capture data from the **FormTextInput**, a UI component developed by the author. This captured data is then sent to the **onSubmit** function when the user clicks the **SubmitFormButton**. The **onSubmit** function, in turn, sends the data to the **createProject** endpoint, which forwards it to the backend for further processing. More Details of the data capture script are defined in the Appendix A.8.

### 5.3.3 Generators from Group 1: Project, Coating, Machine, and Process Parameters

These generators are grouped together as they are the simplest ones. Each generator consists of a single form with options for creating a new entry. The user can input tool data using text inputs and select options from drop-down menus, which gather information from the project or machine generators. The Figures 15 and 16 shows the UI from the 4 generators.

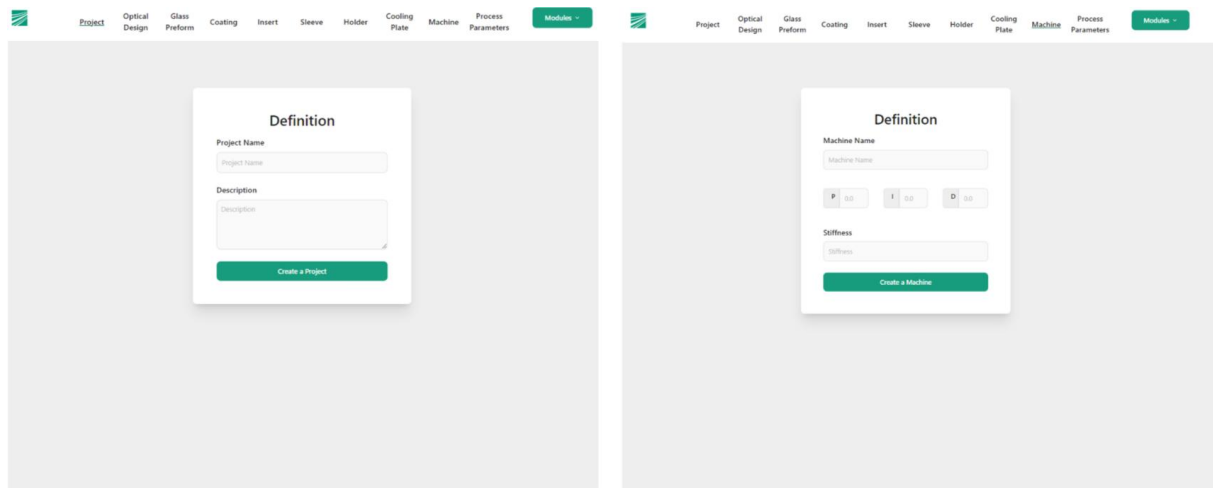


Figure 15 – UI from Project and Machine generators <sup>1</sup>.

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

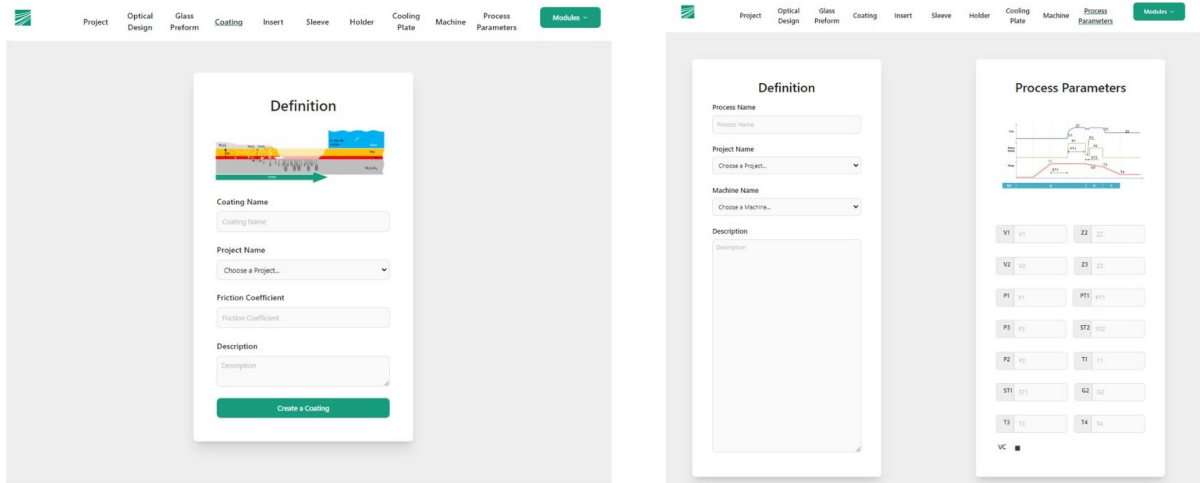


Figure 16 – UI from Coating and Process Parameters generators. <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

The inputs in the Coating and Process Parameters generators use drop-down, which makes them display data from the backend from existing projects or machines in the database. This is done by making a GET request to the backend such that the data is retrieved and displayed for the user. In order to prevent the user from experiencing a delay, the fetching is done by using the **useEffect** hook from ReactJS when the page is rendered. In the following example, we can see the usage of the **useEffect** hook in the Coating page. This hook is used to call the **fetchData** function to make a GET request on all projects to ensure that all drop-down values are populated before the user interacts with them, thus minimizing user-experience delay.

```

1  const [projectDropDownElements, setProjectDropDownElements] = useState
    ([]) // UseState responsible to store and set the values displayed in
        the Drop-Down
2  useEffect(() => {
3      // Code Logic Implemented in the hook
4      const fetchData = async () => {
5          const projects = await getProjects().then(
6              response => {
7
8                  return response.data
9              }
10             ).catch(
11                 (error) => { console.log(error) }
12             )
13             setProjectDropDownElements(projects)
14         };
15         // Optional returning function
16         fetchData();

```

```
17     }, [] // Array Dependency to define the useeffect of listen to a  
        specific state or variable. If the Dependency Array is empty [] will  
        only run once.  
18     );
```

Listing 5.7 – UseEffect in the coating generator responsible for the project

### 5.3.4 Generators from Group 2: Sleeve, Holder, and Cooling Plate

#### 5.3.4.1 PDF Visualizer Component

These three generators are grouped together because they introduce an extra layer of complexity with the **PDFVisualizer** Component. Despite this complexity, they share similar features and functionality with Group 1 generators, albeit with the addition of the **PDFVisualizer** Component.

The **PDFVisualizer** Component is specifically designed to display PDF files, particularly showcasing tool drawings. It offers several key features, including:

- PDF Rendering: Renders PDF files directly within the web application.
- Interactive Viewer: Allows users to zoom in and out, navigate through pages, and view detailed drawings of tools.
- Responsive Design: Styled with Tailwind CSS to ensure a quality and responsive viewing experience across different devices.
- Store the PDF: Implement a Effective way to store the data in the database.

To implement the PDF Visualizer component efficiently and simply, we utilized the `<embed>` tag from HTML, avoiding unnecessary complexity. Before delving into the implementation details, it's essential to understand the process of displaying and storing PDFs.

**Displaying the PDF:** To display the user's PDFs, the component requires a file to be displayed, which can either be provided by the user or retrieved from the database. When the file is provided, a function creates an object URL for the PDF file, enabling the application to display the PDF directly in the browser, ensuring a good viewing experience for the user. In the Figure 17 there is the component implementation.

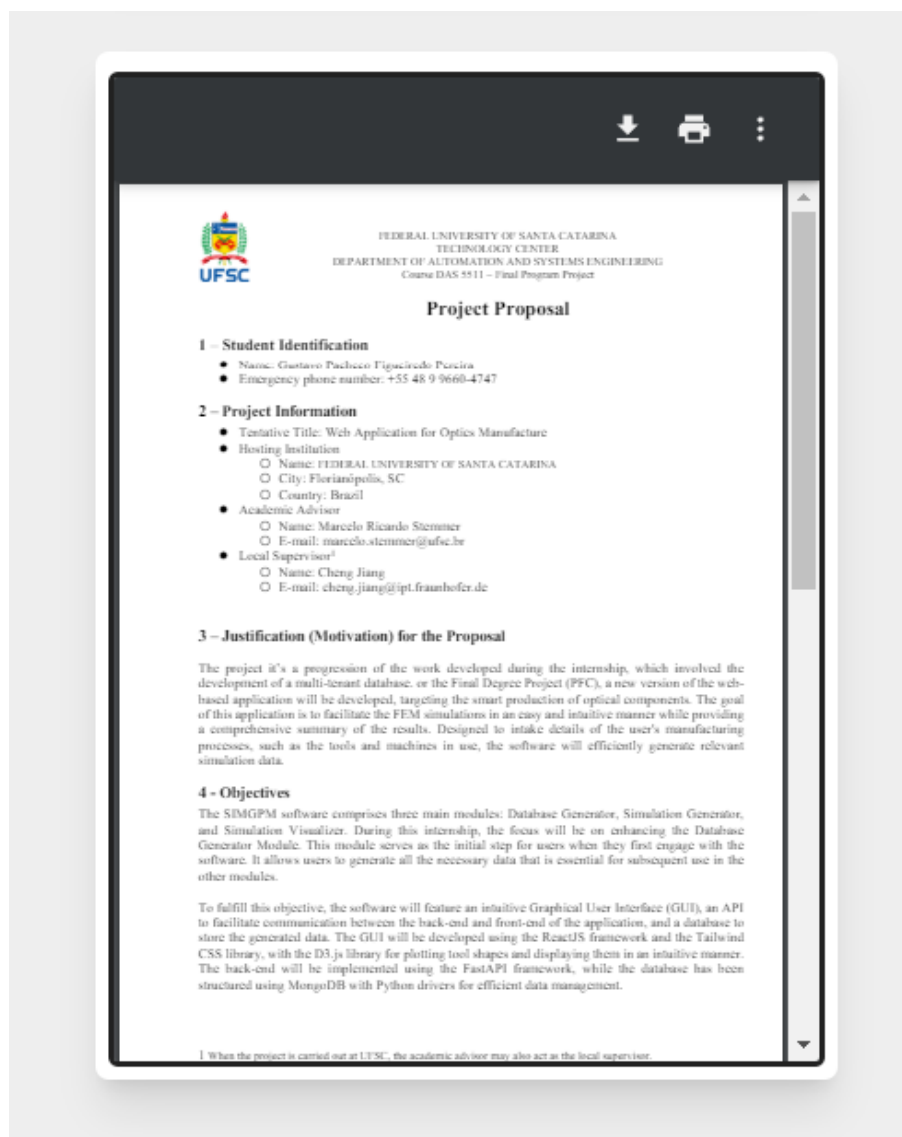


Figure 17 – PDF Visualizer displaying a PDF

**Storing the PDF:** To save the PDF file in the database, it's necessary to convert the PDF to a binary string, allowing for further processing such as extracting data for visualization or performing database operations.

The functions responsible for converting between the binary and string representations are in the Appendix A.9.

In the Figure 18 we can have a better understanding of the PDF visualizer.

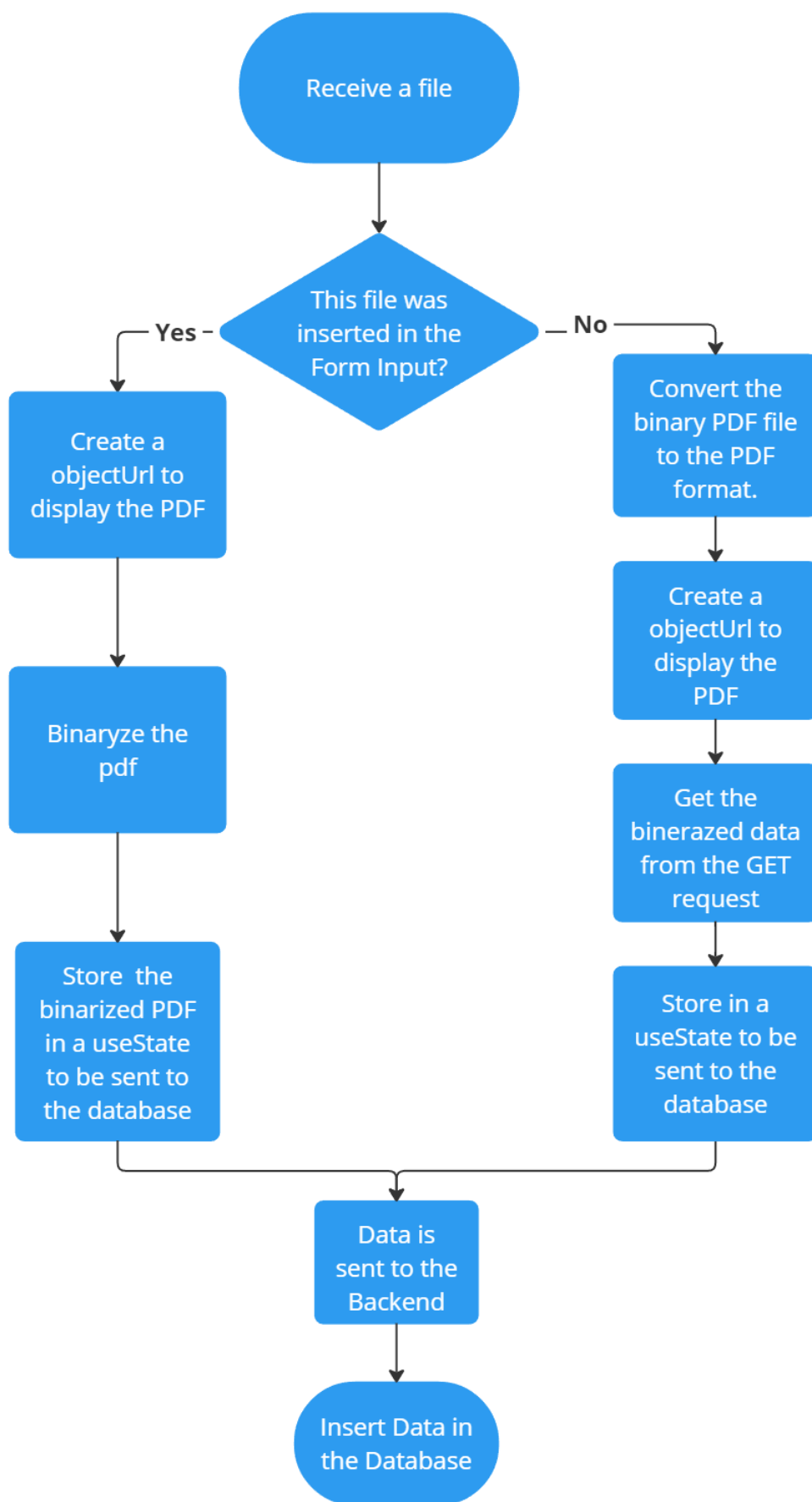


Figure 18 – PDF Visualizer Logic flow chart

### 5.3.4.2 Drop-down Data

All generators from Groups 2, 3, and 4 feature at least two drop-downs. This necessitates multiple data requests to retrieve and display data to the user.

In the SIMPGM project, we implemented a multi-tenant architecture to support multiple users and their respective data within the same application. A critical aspect of this architecture is verifying the origin of each request to ensure that data is securely and correctly associated with the requesting user. However, this verification process introduced significant delays when fetching data for the project and material drop-downs, as each request had to be individually authenticated and processed, which is illustrated in the Figure 19.

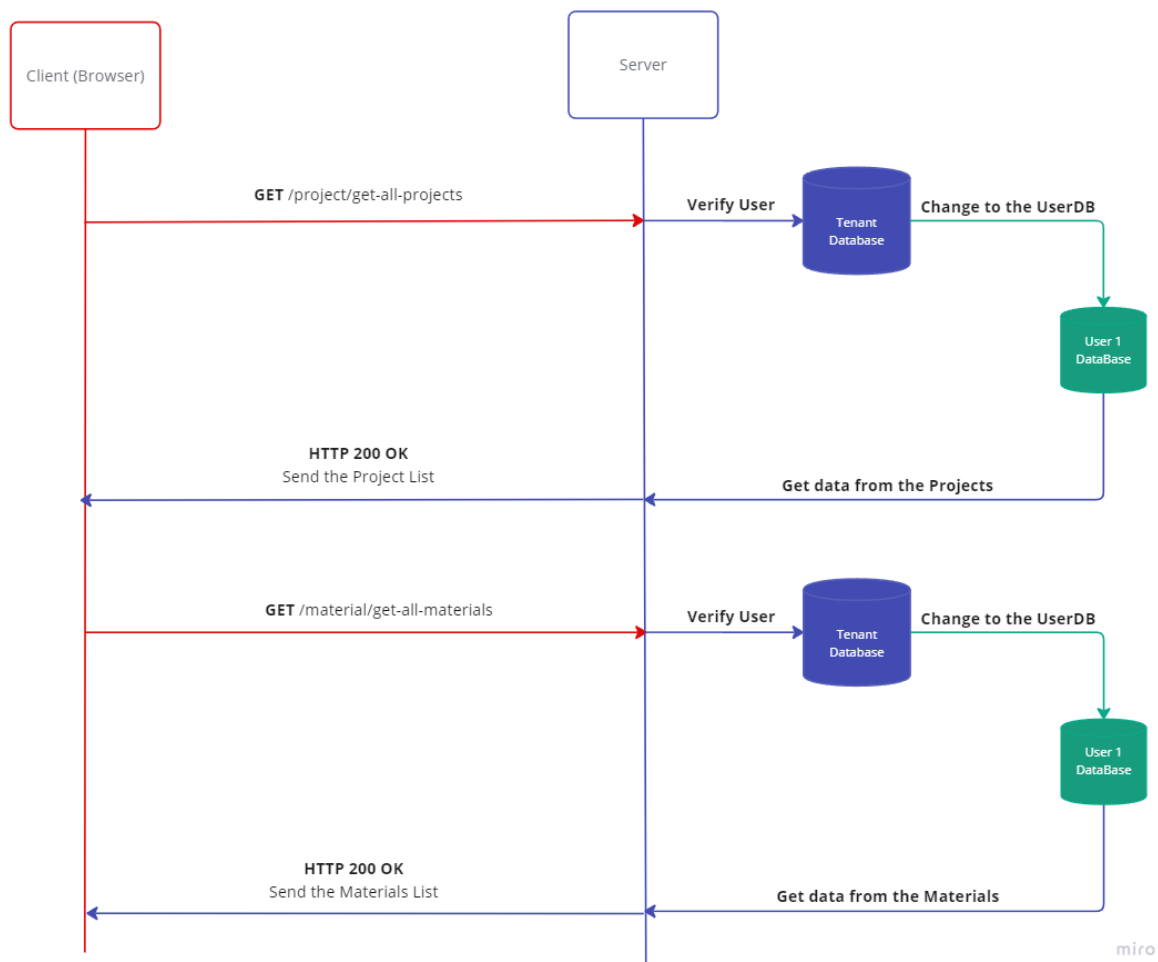


Figure 19 – API diagram with the Request/Response with the previous version approach for the Drop-down data

To overcome this challenge, we developed a dedicated service that consolidates all data operations into a single request, thereby reducing the number of individual requests and minimizing the associated delays. The new implementation logic can be seen in the Figure 20. The Code from the previous and the new approach are in

## Appendix A.10.

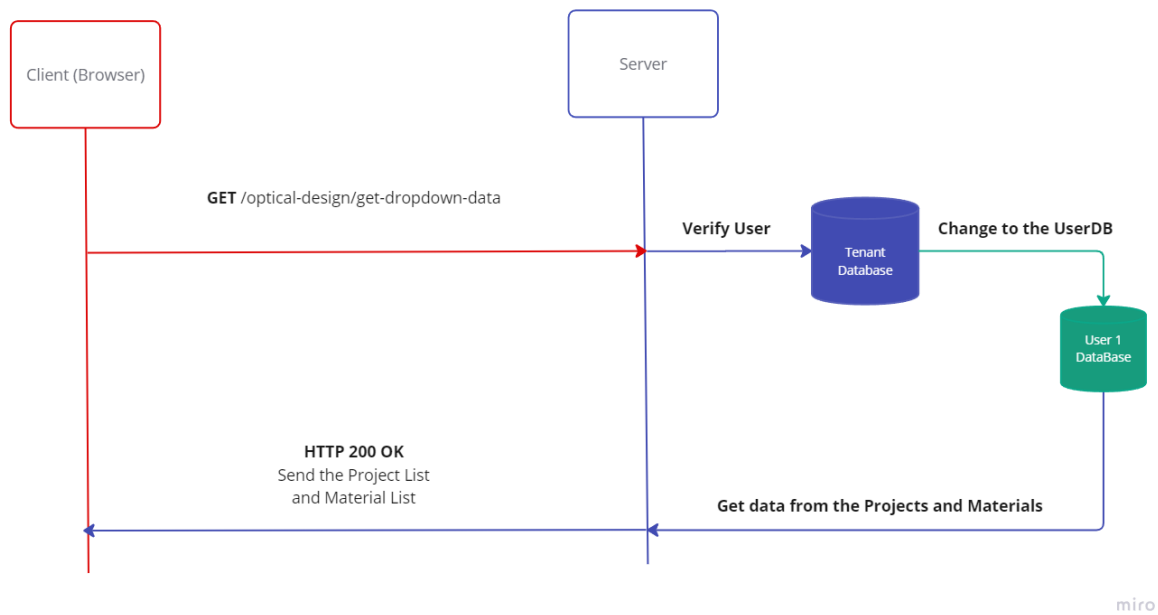
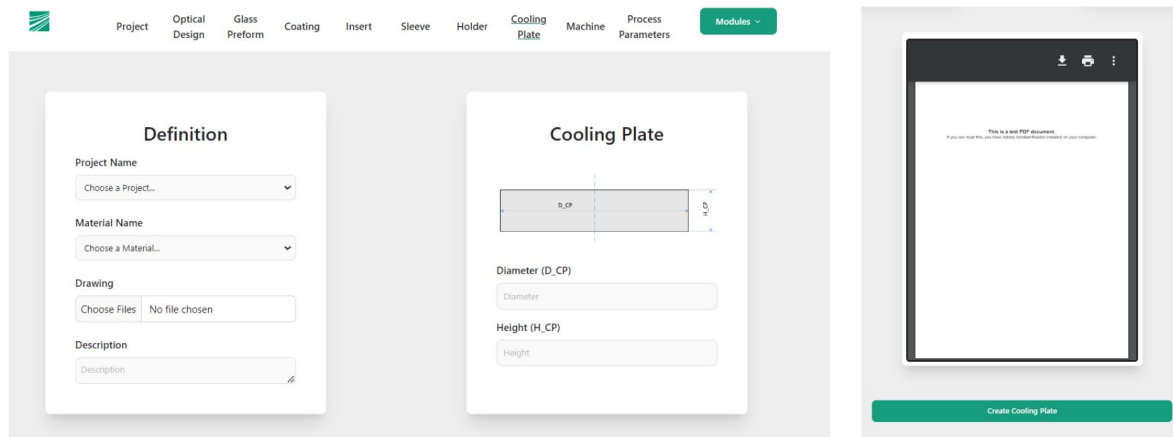


Figure 20 – API diagram with the Request/Response with the new version approach for the Drop-down data

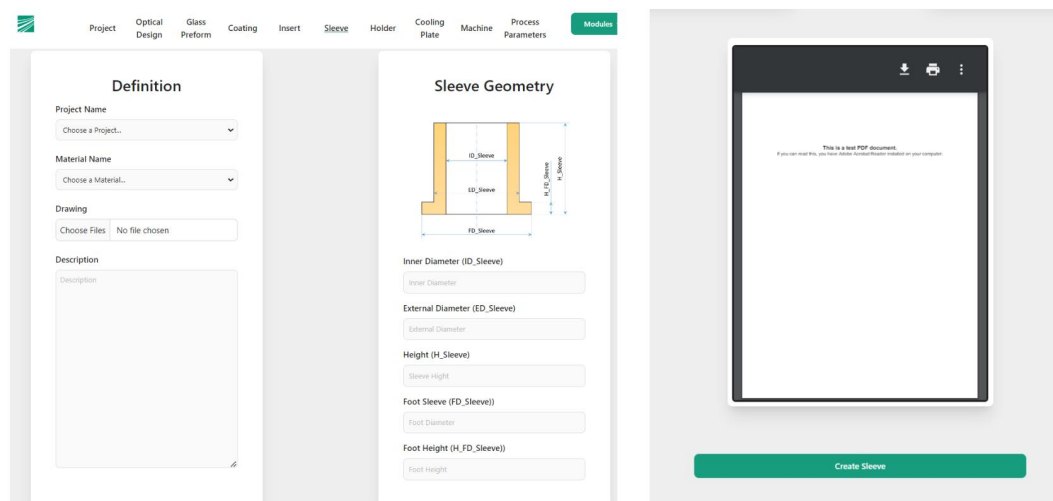
### 5.3.4.3 UI

In this subsection is possible to the developed UI components in the Figures 21, 22, and 23:



Figure 21 – UI from Cooling Plate generator <sup>1</sup>.

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

Figure 22 – UI from Sleeve generator <sup>1</sup>.

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

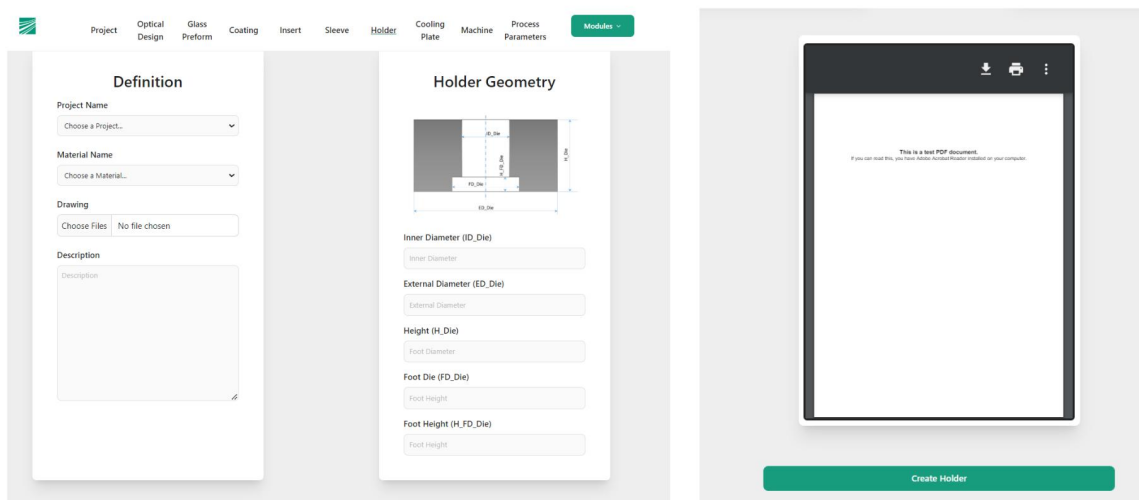


Figure 23 – UI from Holder generator <sup>1</sup>.

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

### 5.3.5 Group 3: Glass Preform

This group introduces the Glass Preform generator, which adds a new component called the Preview Plot. The Glass Preform generator brings an extra layer of complexity due to the possibility of creating eight different types of glass preforms: Ball, Disc, Gob, Bi-convex, Bi-concave, Plano-convex, Plano-concave, and Meniscus.

#### 5.3.5.1 Preview Plot Component

To implement this functionality, an isolated endpoint was developed, along with a dedicated service that has access to the plot algorithms developed by *Cheng Jiang, M.Sc. M.Eng* and Fraunhofer IPT intellectual property. These algorithms are in binary code, and the author does not have direct access to them; instead, the methods inside the Python functions are utilized under the supervision of the author. The Preview Plot is also included in the Group 4 generator. What makes this component complex is that each glass type has a different plot.

In the user interface, there is a drop-down menu where the user can select the preform they want to work with. Once selected, the UI displays the preform drawing with fields for the user to insert parameters specific to the type of preform. This Process is more clear in the API diagram in the Figure 24 and in the flowchart in the Figure 25. More details about the code can be seen in the Appendix A.11

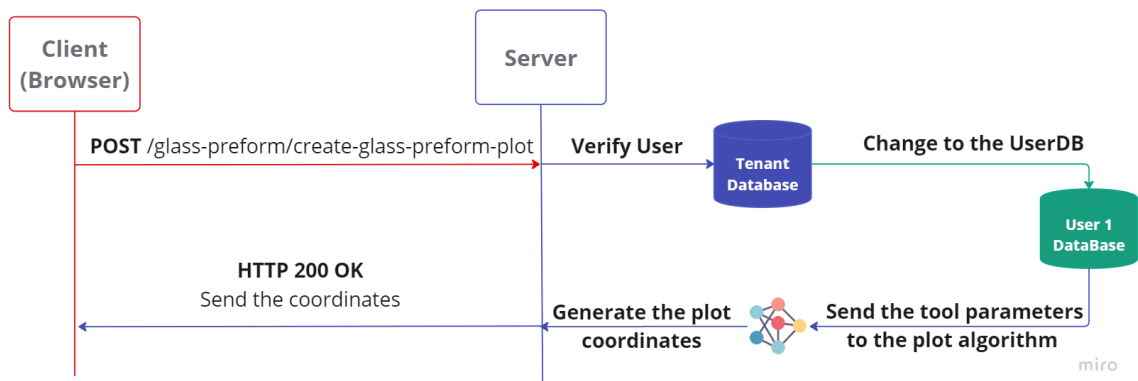


Figure 24 – API diagram with the Request/Response with the plot generation logic

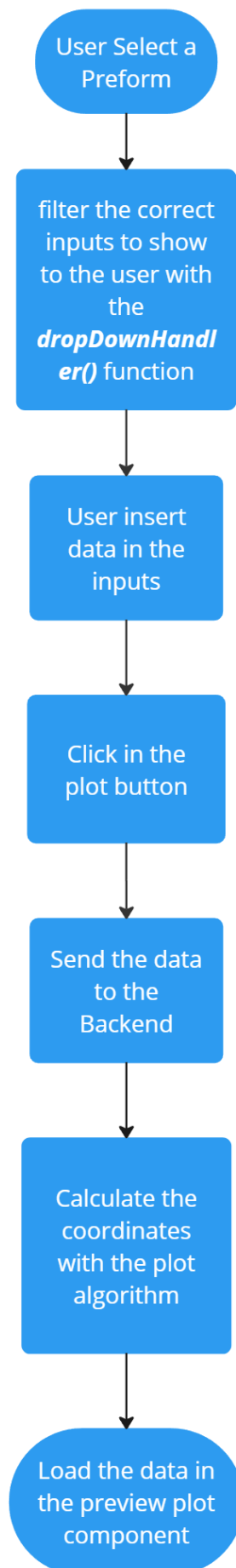
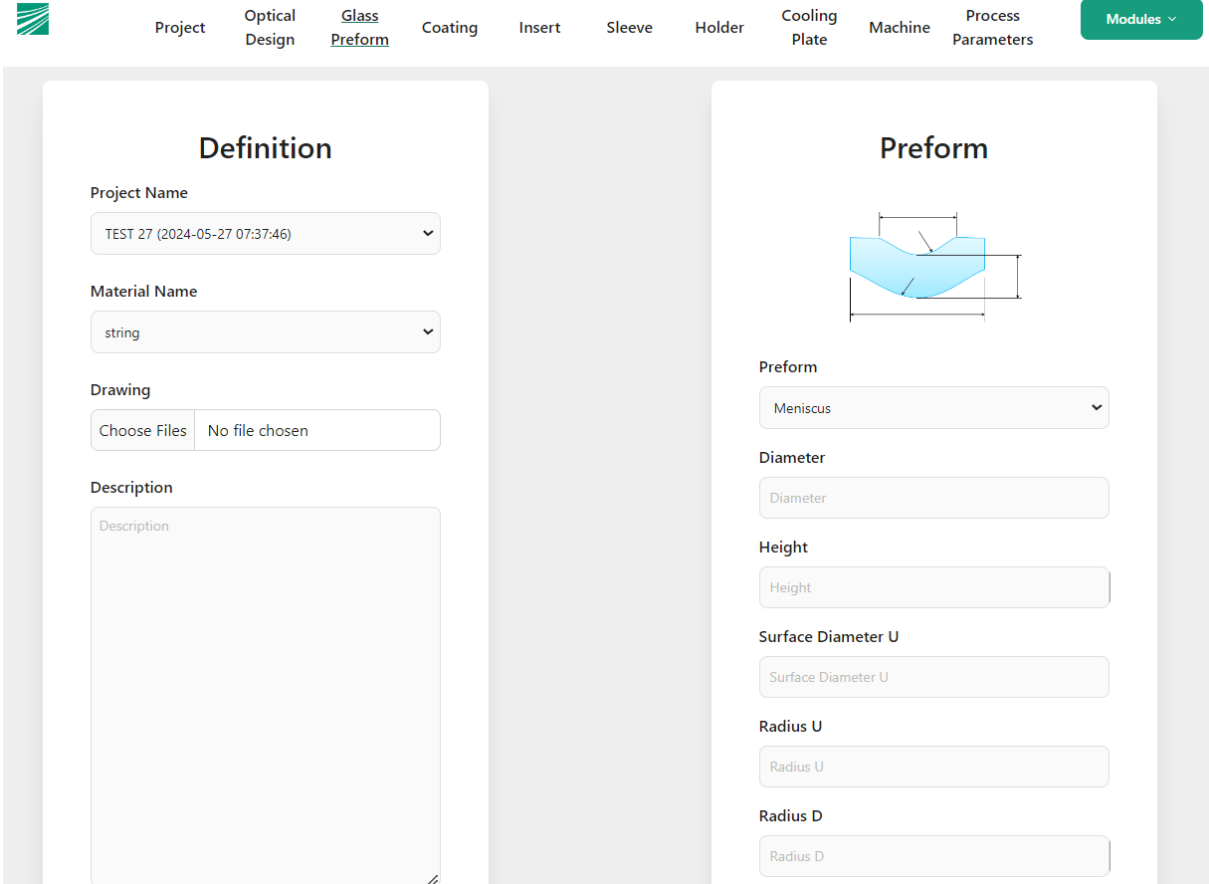


Figure 25 – Preview Plot Flowchart

### 5.3.5.2 Glass Preform UI

In this subsection is possible to the developed UI components in the Figures 26, 27, 28, and 29:



The screenshot displays a web application interface for defining a glass preform. The top navigation bar includes a logo on the left and menu items: Project, Optical Design, Glass Preform (highlighted), Coating, Insert, Sleeve, Holder, Cooling Plate, Machine, Process Parameters, and a Modules dropdown. The main content area is split into two panels. The left panel, titled 'Definition', contains fields for Project Name (TEST 27 (2024-05-27 07:37:46)), Material Name (string), Drawing (Choose Files, No file chosen), and a Description text area. The right panel, titled 'Preform', features a diagram of a blue meniscus-shaped preform with dimension lines. Below the diagram are input fields for Preform (Meniscus), Diameter, Height, Surface Diameter U, Radius U, and Radius D.

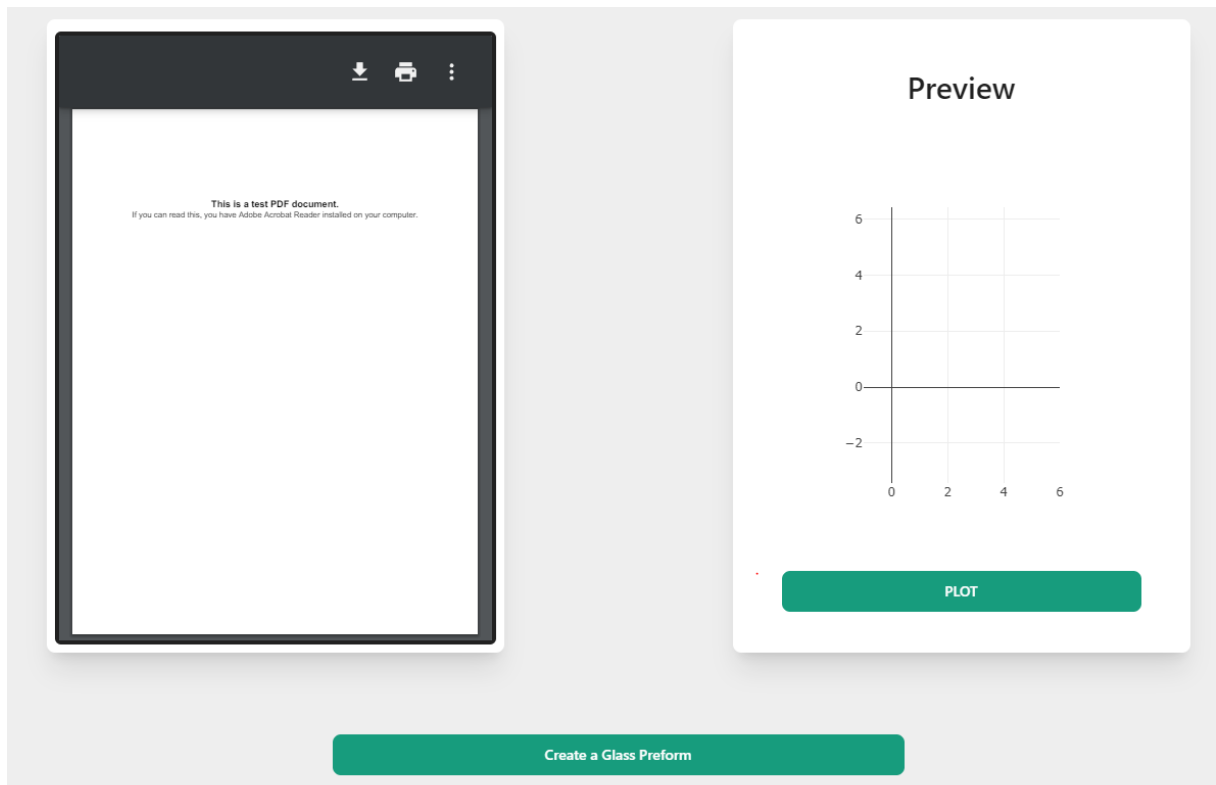
Figure 26 – Glass Preform UI with Meniscus preform Selection <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

The screenshot displays a web application interface for defining a glass preform. The top navigation bar includes a logo on the left and menu items: Project, Optical Design, Glass Preform (highlighted), Coating, Insert, Sleeve, Holder, Cooling Plate, Machine, Process Parameters, and a Modules dropdown. The main content area is split into two panels. The left panel, titled 'Definition', contains four sections: 'Project Name' with a dropdown menu showing 'TEST 27 (2024-05-27 07:37:46)'; 'Material Name' with a dropdown menu showing 'string'; 'Drawing' with a file selection area showing 'Choose Files' and 'No file chosen'; and 'Description' with a text input field containing the placeholder 'Description'. The right panel, titled 'Preform', features a 3D visualization of a light blue sphere with dimension lines indicating its diameter. Below the visualization are two dropdown menus: 'Preform' with 'Ball' selected, and 'Radius' with 'Radius' selected.

Figure 27 – Glass Preform UI with Ball preform Slection <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

Figure 28 – Glass Preform UI <sup>1</sup>

---

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

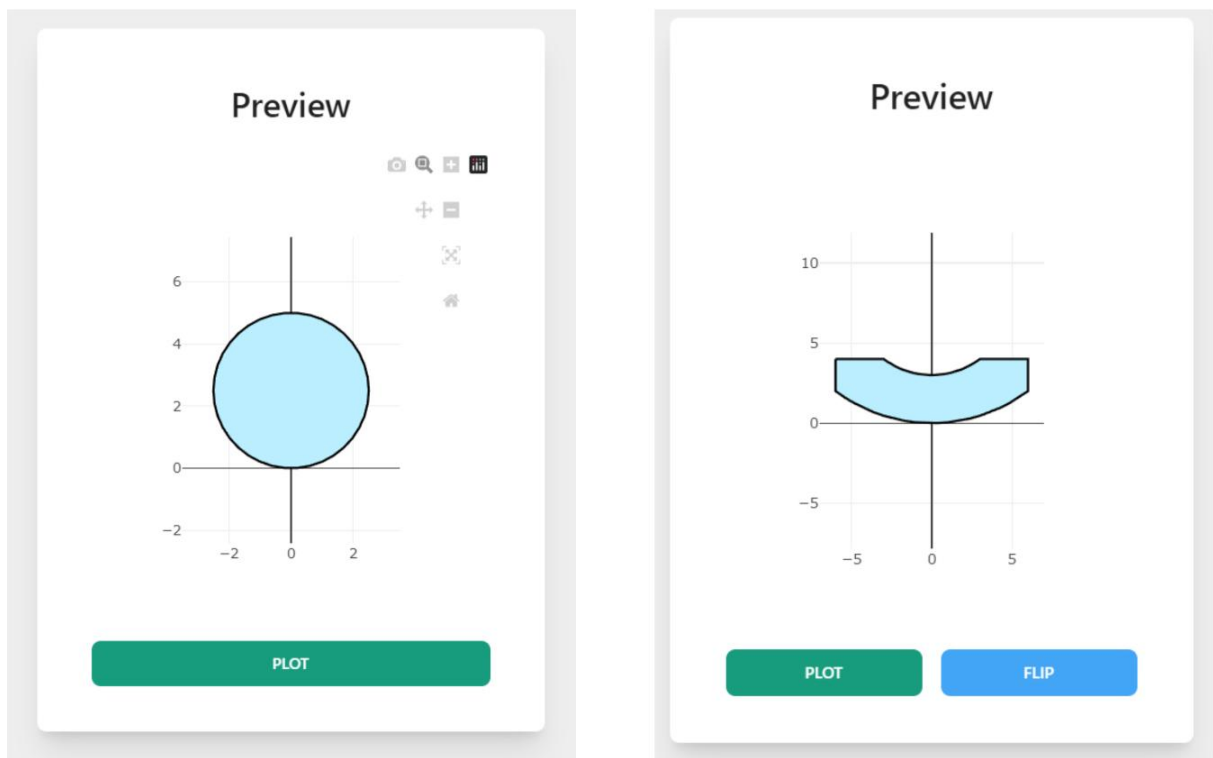


Figure 29 – Glass Preform with Ball and Meniscus Plot <sup>1</sup>

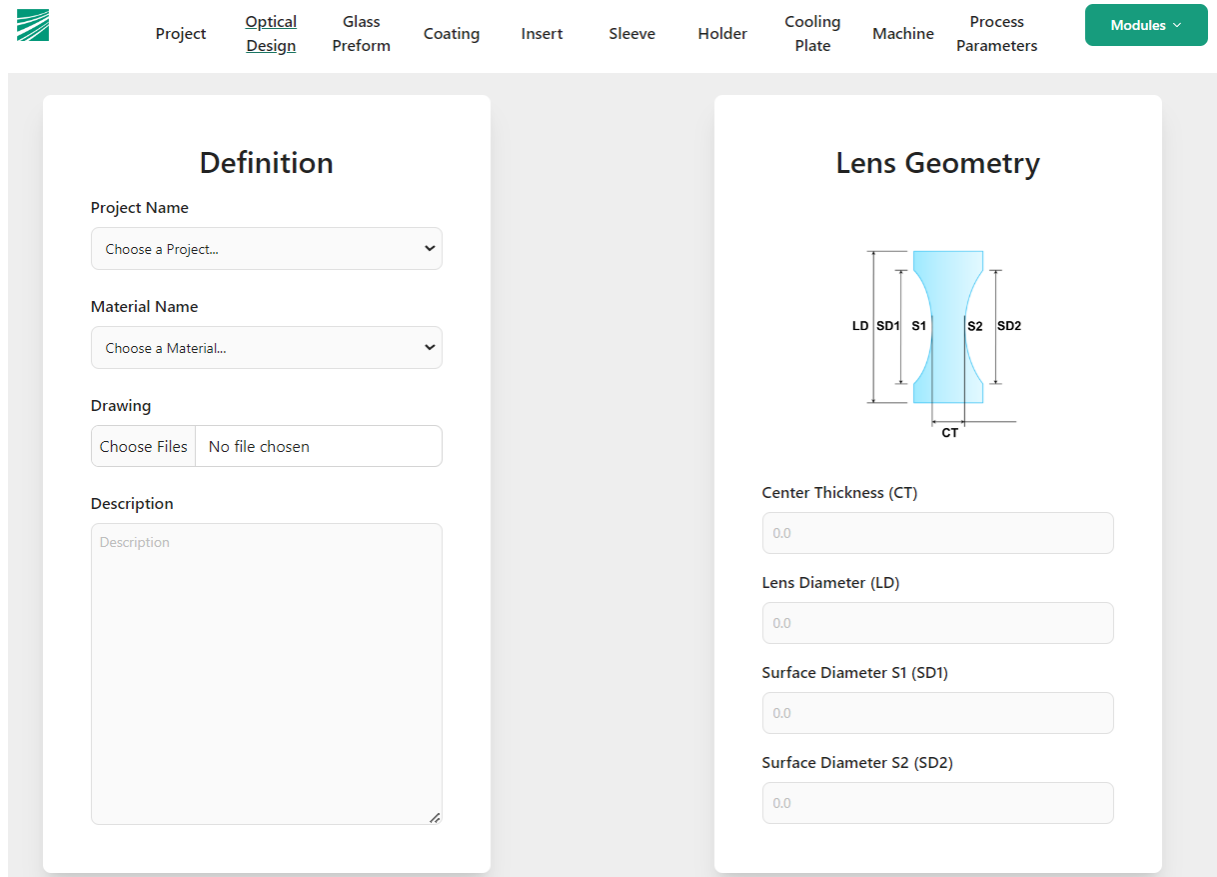
<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

### 5.3.6 Group 4: Optical Design and Insert

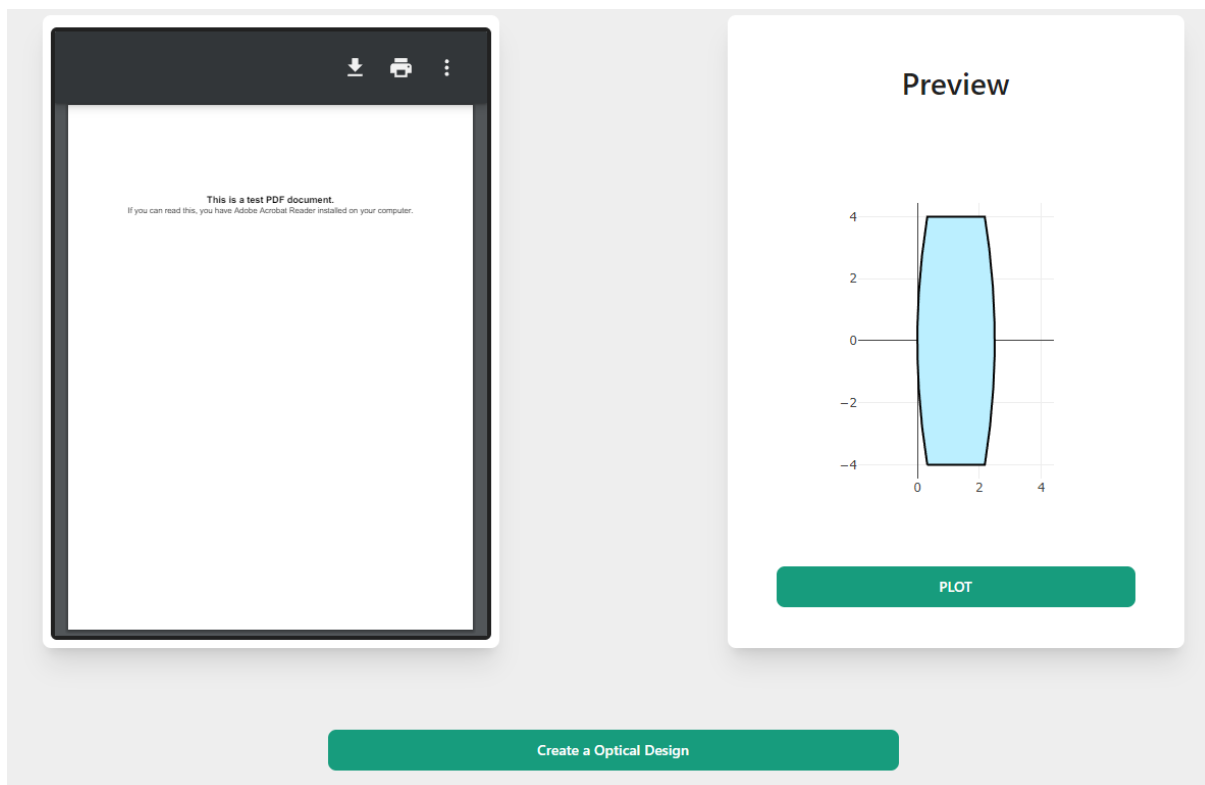
These generators are grouped together due to their complexity, as they introduce several advanced components such as Modals, Chamfer Tables, Sag Tables, and Preview Plots. Both the Insert and Optical Design generators feature similar user interfaces, which include a definition and geometry form, a preview plot, and a PDF visualizer.

The new component, the Modal, is activated when the user clicks on the image within the geometry form. This action opens a modal where the user can input detailed information about the surfaces. While each Modal in the Insert and Optical Design generators has its unique features, they both include sections for surface inputs, Chamfer Table, Sag Table, and Preview Plot functionalities. These consistent sections ensure a streamlined user experience, allowing for precise and efficient data entry and visualization. Below we can see the Optical Design UI in the Figures 30 and 31.



Figure 30 – Optical design UI <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

Figure 31 – Optical Design UI <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

### 5.3.6.1 Chamfer Table

The Chamfer Table component is the most complex feature of the module, enabling users to precisely define chamfers for surfaces. This component comprises a responsive table where users can input chamfer details, with options for both slope and round chamfers.

When opting for the slope option, users are prompted to specify the length and angle of the chamfer. Conversely, selecting the round option requires only the radius input. Users can add chamfers iteratively until the cumulative length does not exceed the surface diameter. If the cumulative length of the chamfers is equal to the surface diameter, the chamfer table should be empty, and the value displayed should be None.

Several additional functionalities have been incorporated into the Chamfer Table to enhance user experience:

- **Preventing Adjacent Rounded Chamfers:** The system does not allow the addition of two rounded chamfers that are adjacent to each other, as this may result in undesired surface configurations.

- **Limiting Length Value:** Users are prohibited from adding a length value greater than the diameter of the surface, as this could lead to geometric inconsistencies.
- **Summation Constraint:** The system ensures that the summation of chamfer lengths does not exceed the diameter of the surface, preventing excessive modification of the surface profile.
- **Angle Calculation:** If the user selects a slope, the system automatically calculates and displays the angle of inclination for the chamfer element, aiding in precise chamfer definition.
- **Preventing Distortion:** Users are prevented from adding values for the radius or length that would result in distortion of the surface geometry, maintaining design integrity.
- **Error Correction:** In case the user inputs an invalid value, the chamfer table calculates a possible corrected value and presents it to the user, allowing for quick resolution of input errors.

To make the implementation of these functionalities possible, an algorithm for the geometric calculation was provided by Fraunhofer IPT and integrated it into the application.

#### 5.3.6.2 Sag Table and Preview Plot Features

The Sag Table and Preview Plot features are designed to provide users with visual and analytical feedback on the data they input. By simply adding a value in the increment input field, users can generate a graph and preview plot, offering immediate visual confirmation and insights into their data.

In the Sag Table, each endpoint of the chamfer is highlighted in red, making it easier for users to identify key points. This visual cue enhances user understanding and facilitates quicker analysis of the data.

The Preview Plot allows users to zoom in and inspect every point of the graph, enabling detailed examination and analysis of the data. This interactive capability empowers users to explore their data comprehensively, facilitating informed decision-making and optimization.

Together, the Sag Table and Preview Plot features provide users with powerful tools for visualizing and analyzing their data, enhancing the usability and effectiveness of the module. In the Figure 32 there is a visual example of the sag table and the plot surface.

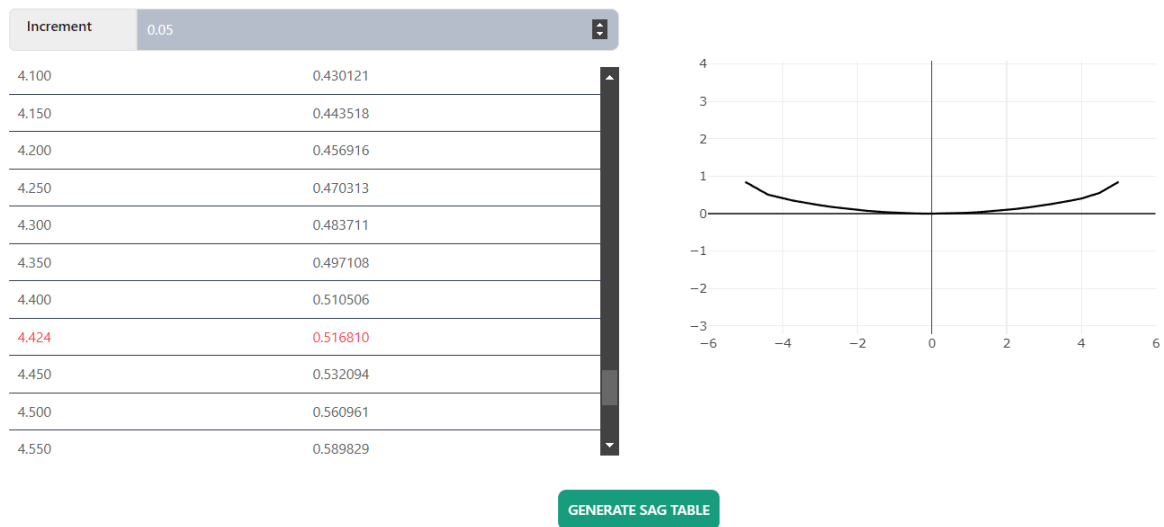


Figure 32 – Sag Table and Preview Plot Feature <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

### 5.3.6.3 Optical Design Surface UI

For the optical design there are sections for surfaces S1 and S2. Also, the sag tables, preview plots, and chamfer tables are dedicated for each surface. These UI elements can be seen in the Figures 33, 34, and 35.

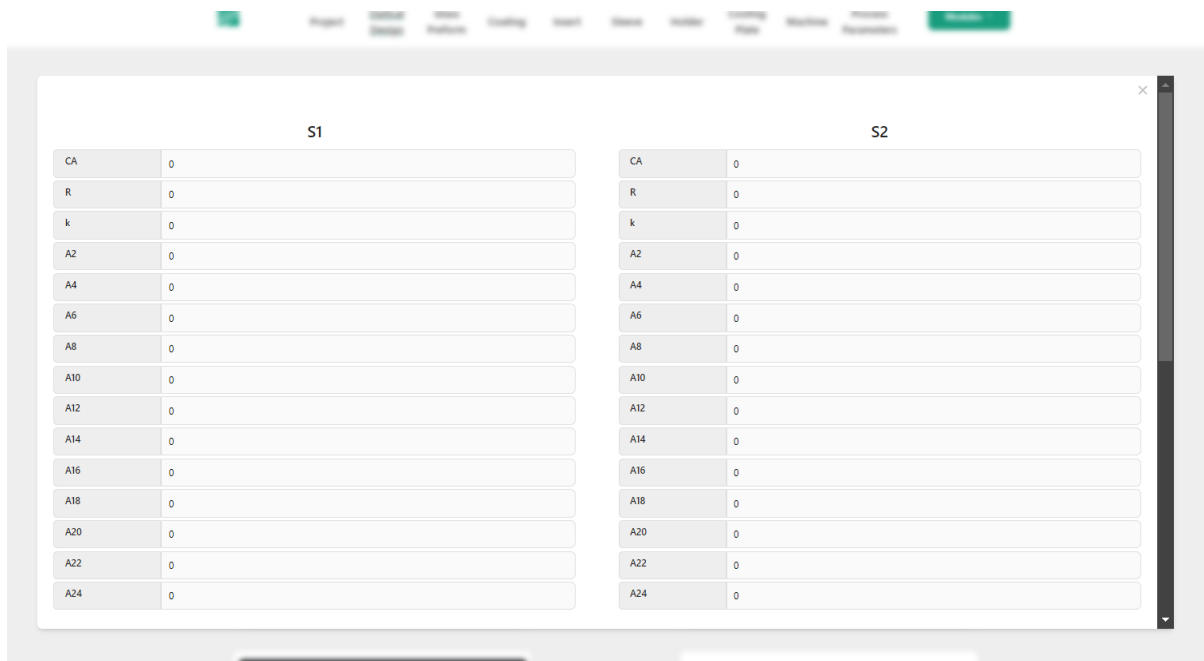


Figure 33 – Optical Design Modal UI <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

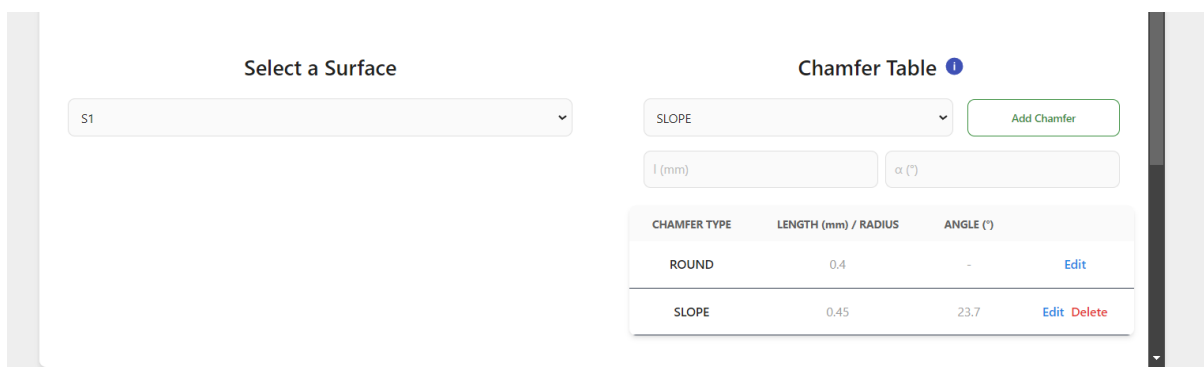
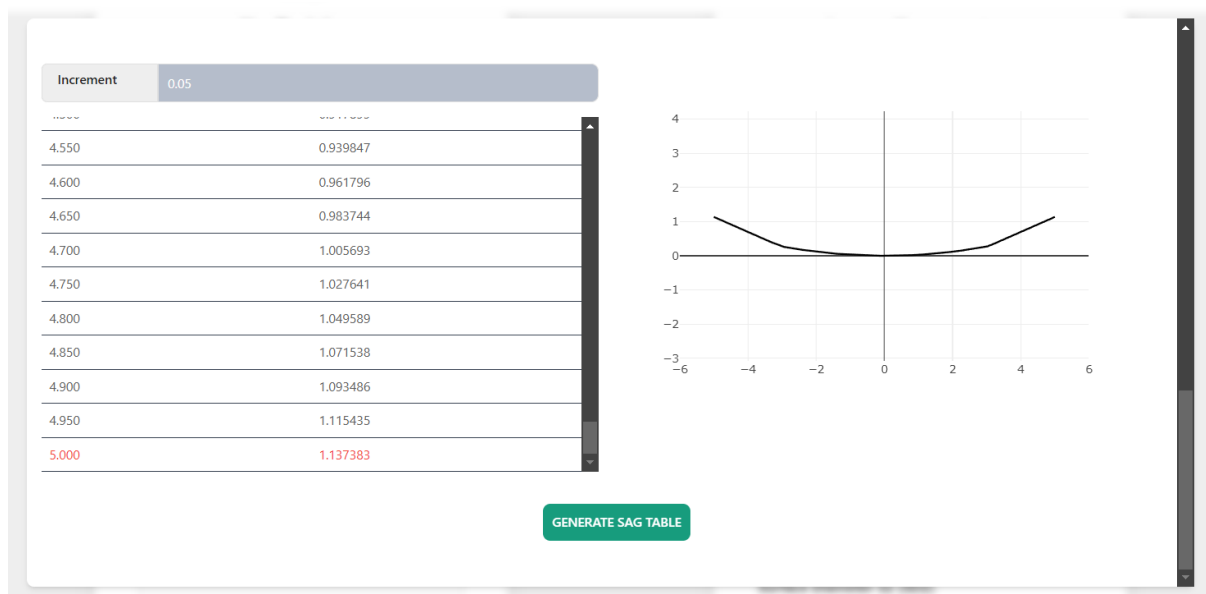


Figure 34 – Optical Design Modal UI <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

Figure 35 – Optical Design Modal UI <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

#### 5.3.6.4 Insert UI

The user interface for defining the Insert offers two options: Flat and Curvature. Selecting the Flat option disables all additional functionalities and simply displays an image, providing a straightforward view of the insert.

In contrast, selecting the Curvature option reveals several advanced features tailored for detailed surface design. These features include an input field for the surface diameter, a dedicated chamfer table, and a sag table specific to the curved surface.

This setup allows users to switch between a simplified view and a more comprehensive design environment based on their specific needs, ensuring flexibility and ease of use in the optical design process. In the Figures 36, 37, 38, and 39 it's possible to see the UI elements developed and their differences between the curvature and flat surface selection.

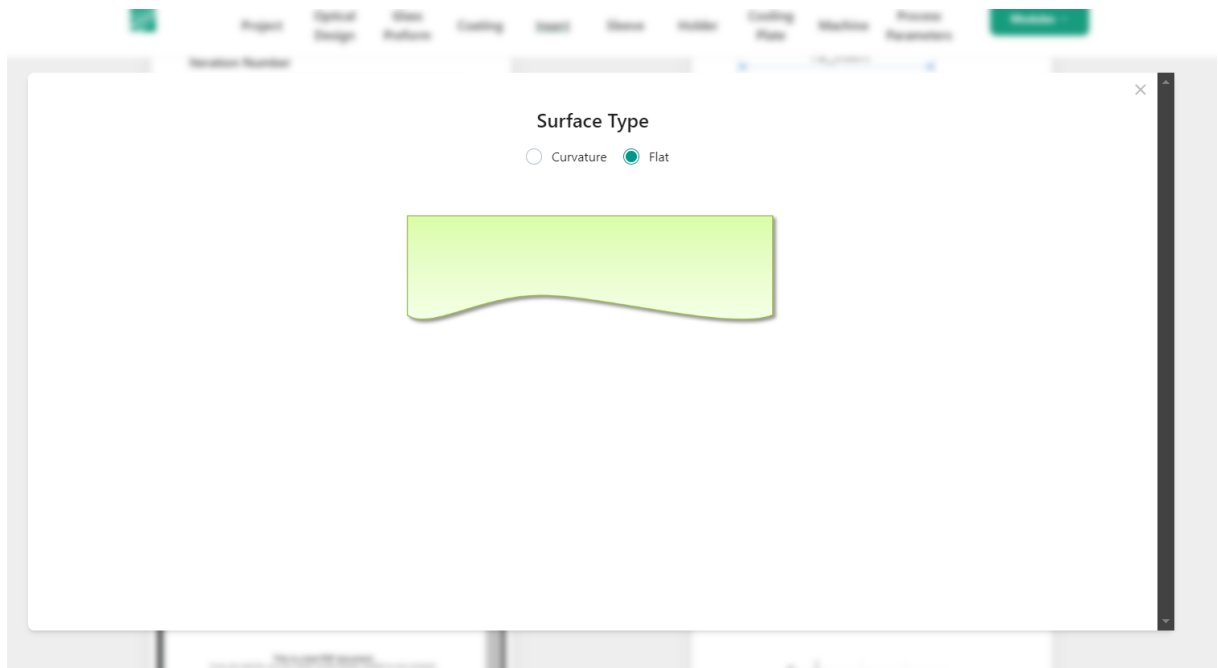


Figure 36 – Insert Modal with Flat Surface Type selected

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

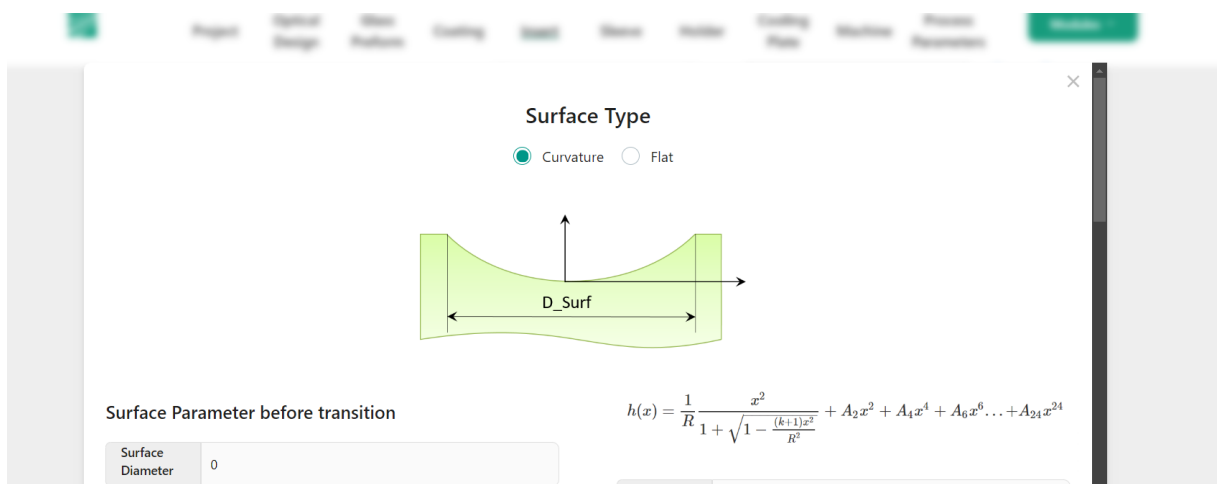
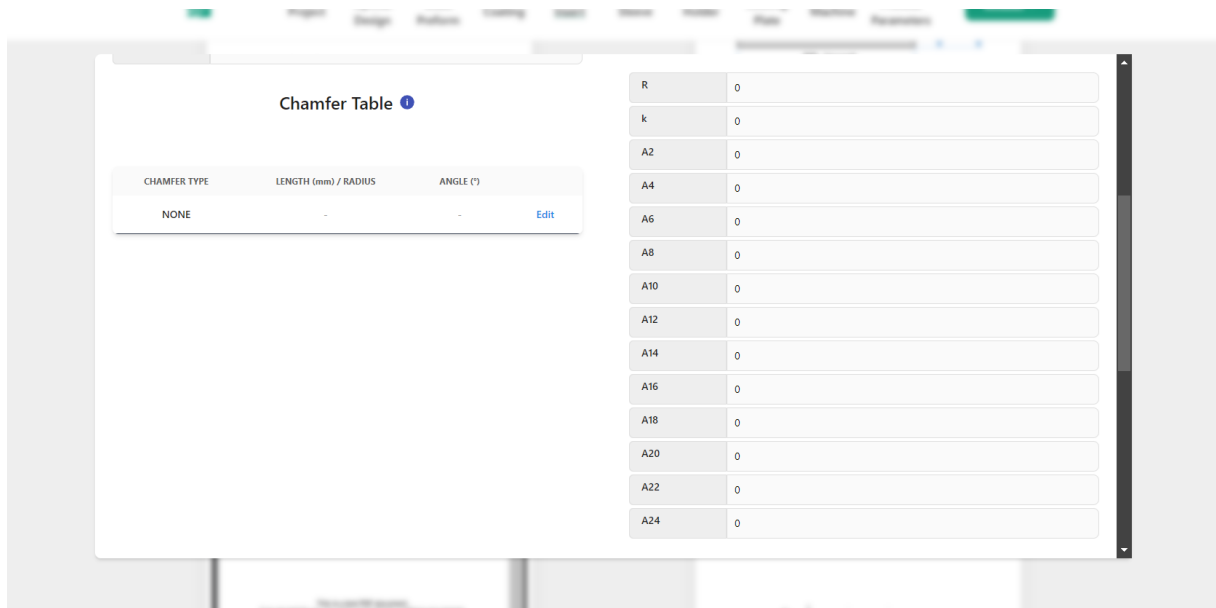
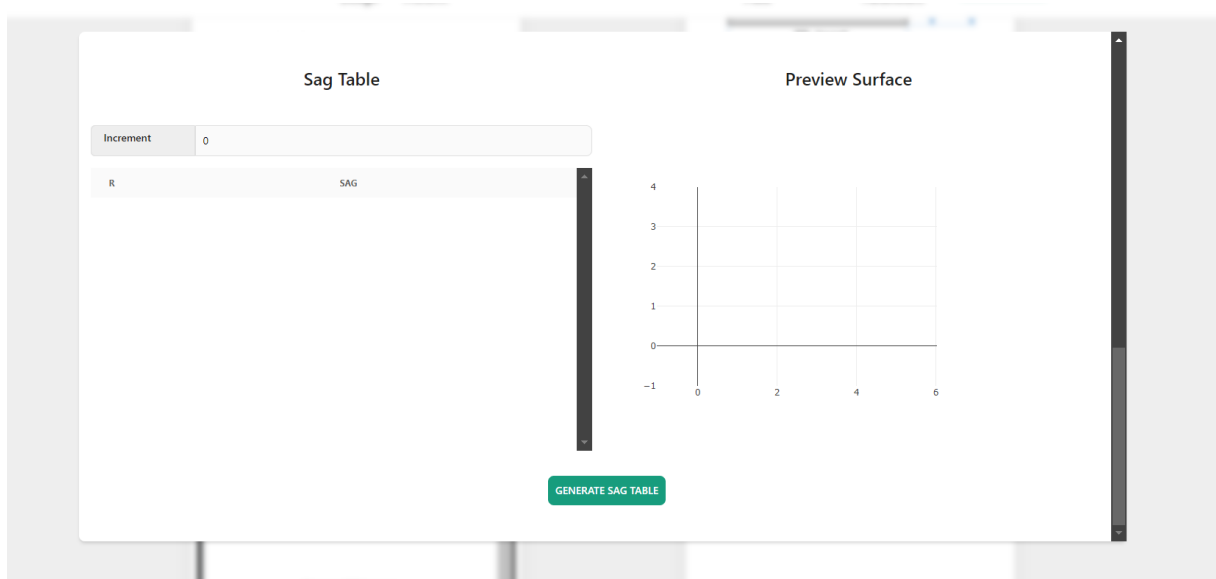


Figure 37 – Insert Modal Curvature <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

Figure 38 – Insert Modal Chamfer Table <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

Figure 39 – Insert Modal Curvature <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

#### 5.4 MOLDING TASK IN SIMULATION MANAGER MODULE

This module is responsible for assembling the data created in the database generator and generating simulation data to be sent to the ABAQUS Generator. The



module consists of two main functions: the Molding Task and the Simulation Generator. However, for this project, only the Molding Task function was implemented.

The Molding Task function involves gathering information from the previous modules. To facilitate this process, a Definition Form was created to collect data such as task name, project name, task number, and concept number. Additionally, a Sidebar was developed to retrieve data from modules such as Optical Design, Process, Machine, and Flange. Lastly, a Modal was implemented to create a Molding concept.

The Definition Form, Sidebar, and Modal collectively streamline the data collection process for the Molding Task, ensuring that all necessary information is gathered efficiently and accurately.

### 5.4.1 Definition Form

The Molding Task definition form the user can add some information about the molding task such as a name, associate to project, give a number and associate a concept number to the task. Which is illustrated in the Figure 40

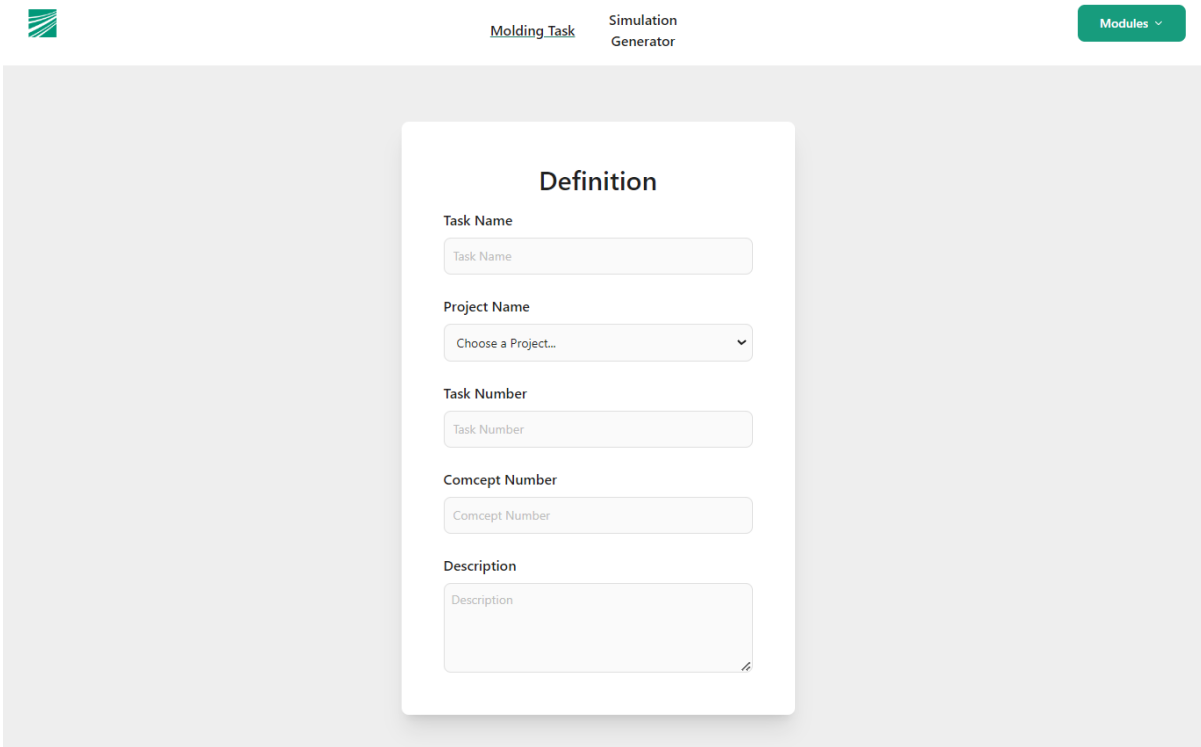


Figure 40 – Molding Task Definition Form <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

## 5.4.2 Molding Task Side Bar

To optimize the user experience, a sidebar was crafted, incorporating a drop-down menu for selecting the appropriate tool. Upon clicking on the card, the sidebar emerges, revealing a drop-down menu showcasing the array of available tools for selection. In the Optical Design segment, the sidebar boasts the inclusion of a drawing PDF, furnishing users with invaluable visual reference material. Similarly, within the Process section, the sidebar presents both the drop-down menu and the process parameters, enabling users to effortlessly select their desired process and input the pertinent parameters. The side bars and Cards developed for the UI can be seen in the Figures 41, 42, and 43

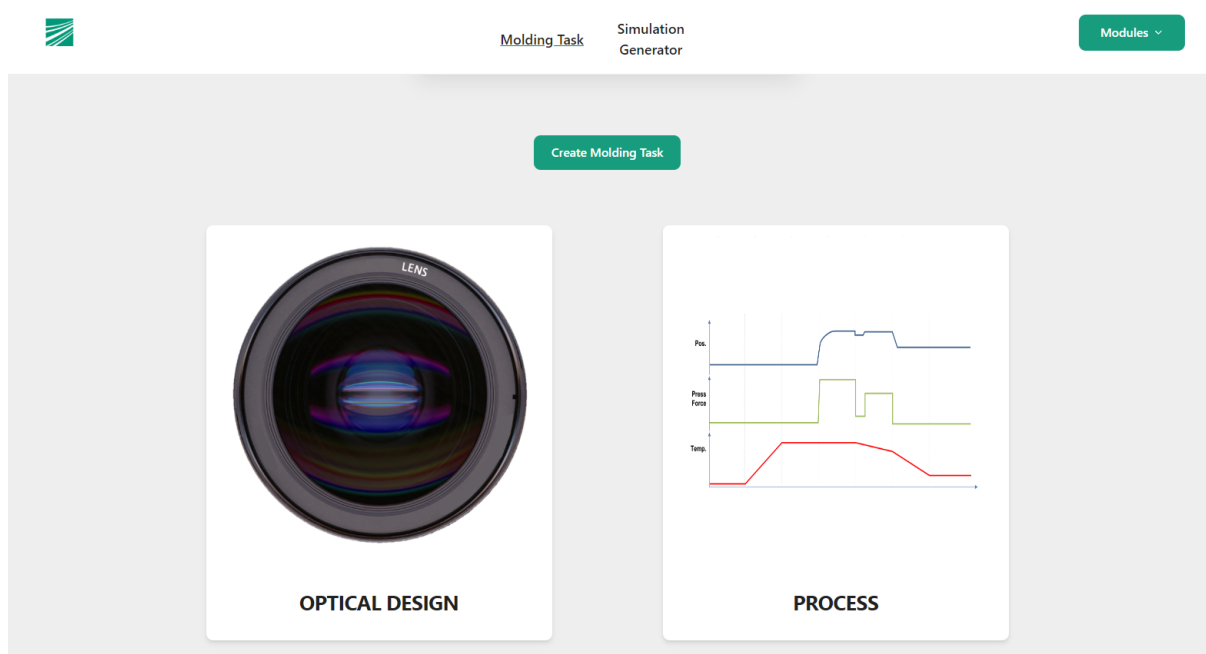


Figure 41 – Optical Design Process cards

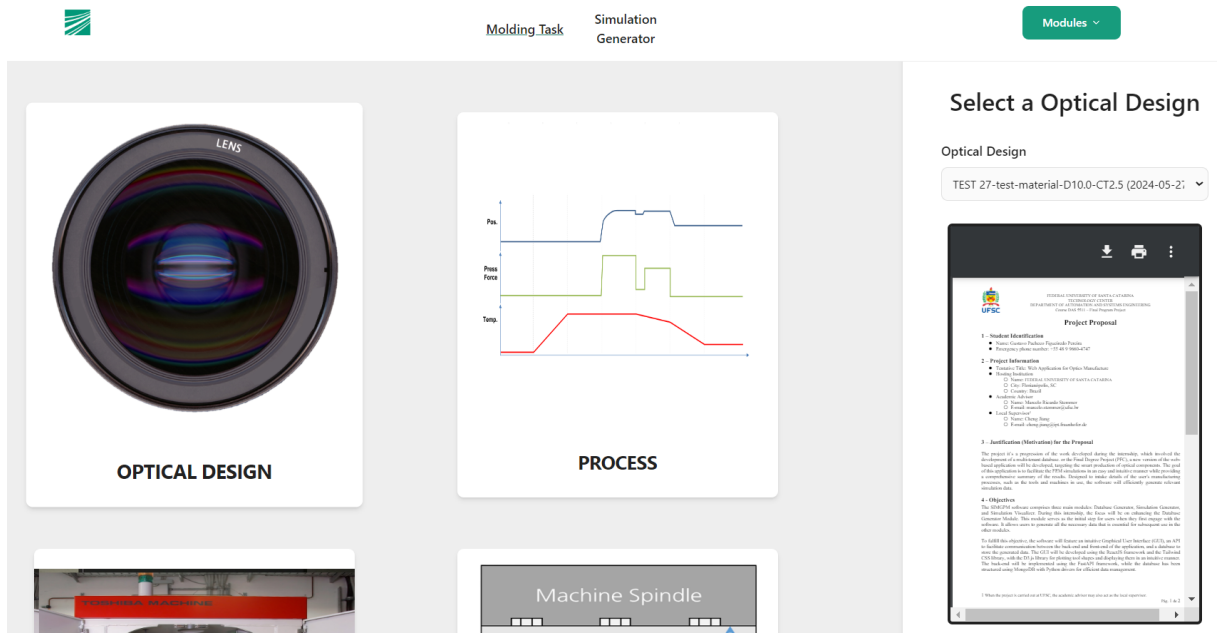


Figure 42 – Optical Design side Bar

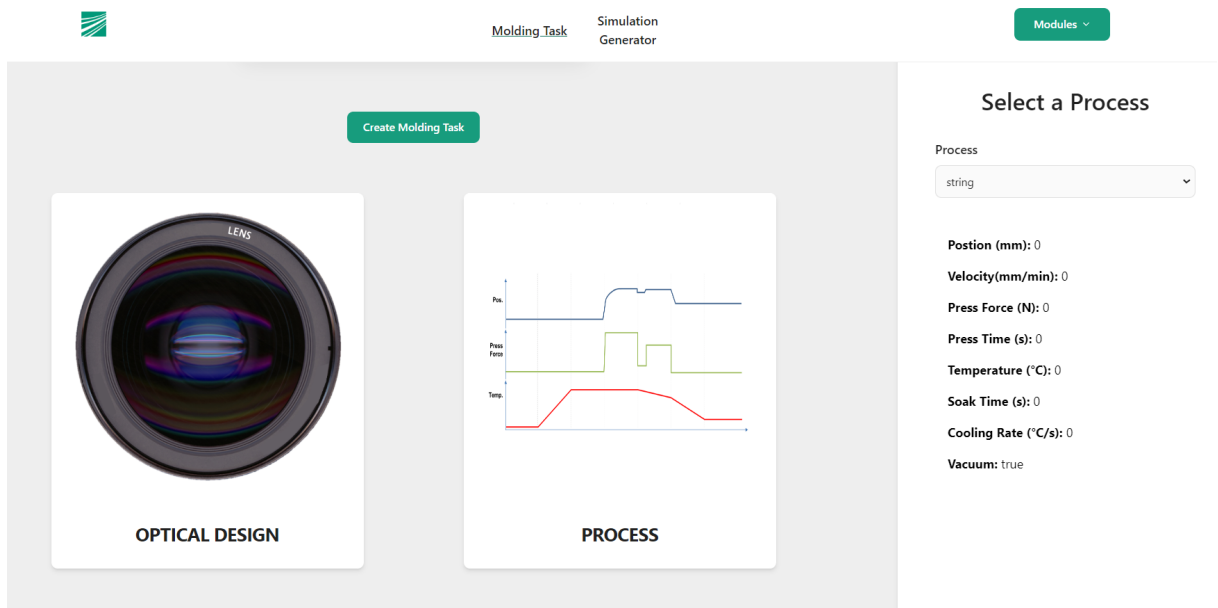
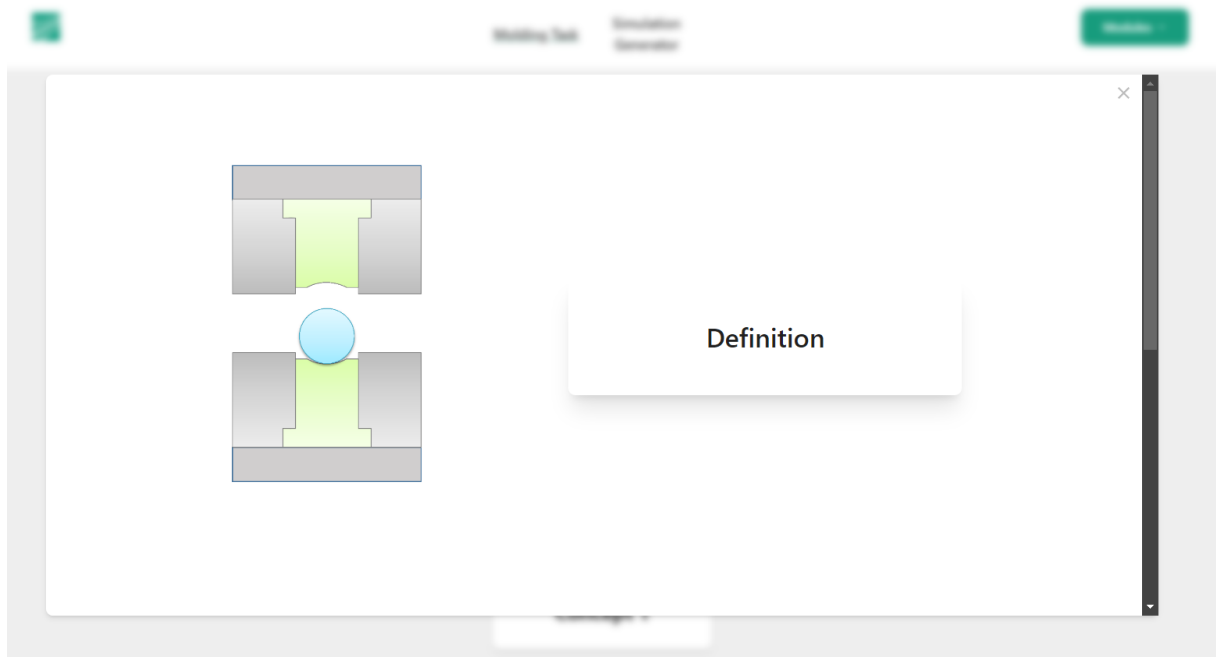


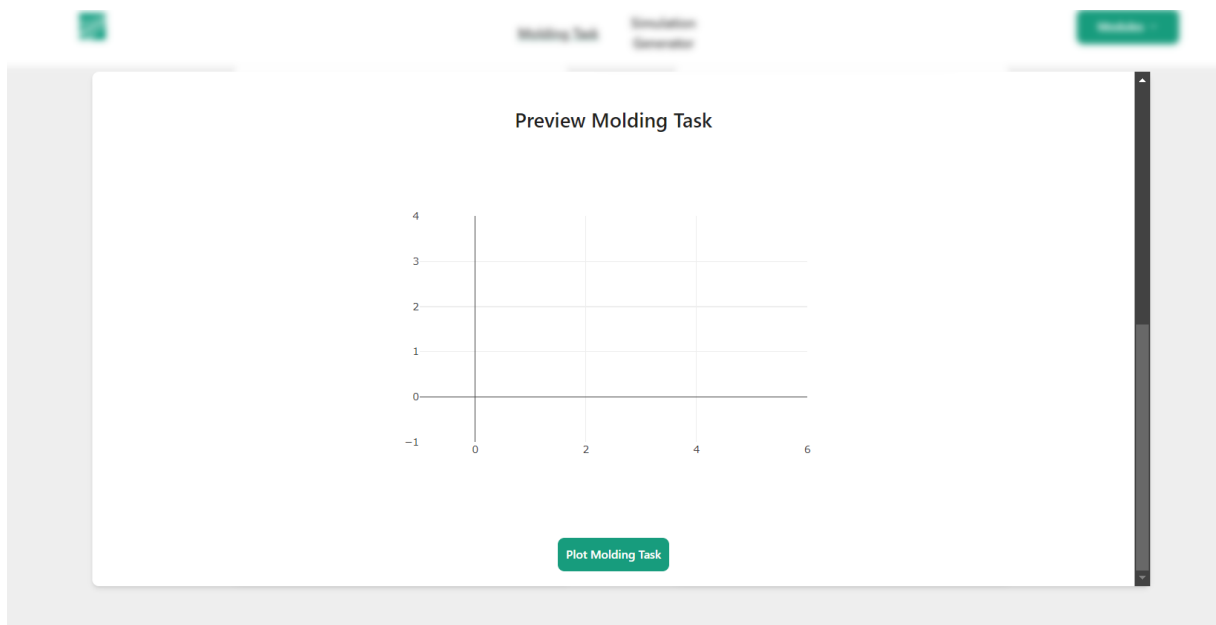
Figure 43 – Process Side Bar

### 5.4.3 Molding Concept

The Molding Concept is encapsulated within a card, which upon user interaction, triggers the opening of a modal. Within this modal, users are greeted with the imagemapper concept, interactive forms, and a preview plot of the molding task. The Modals UI is represented in the Figures 44 and 45.

Figure 44 – Molding Concept Modal <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

Figure 45 – Molding Concept Modal <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

## 5.4.4 ImageMapper

### 5.4.4.1 Overview of react-img-mapper

The react-img-mapper library allows you to create responsive image maps in a React application. It provides a component that takes an image and a set of coordinates defining clickable areas on that image. These areas can trigger events such as clicks, hovers, and more.

### 5.4.4.2 Image Mapper

The image mapper component is integrated into the application, serving as a tool for users to interact with graphical representations of their molding tasks. When users click on specific areas of the image, corresponding forms are presented, enabling them to select the appropriate tools necessary for their task. This intuitive functionality, powered by react-img-mapper, plays an important role in simplifying tool selection, thereby enhancing the overall user experience. In the Figures 46 and 47, there are the Image Mapper implementation.

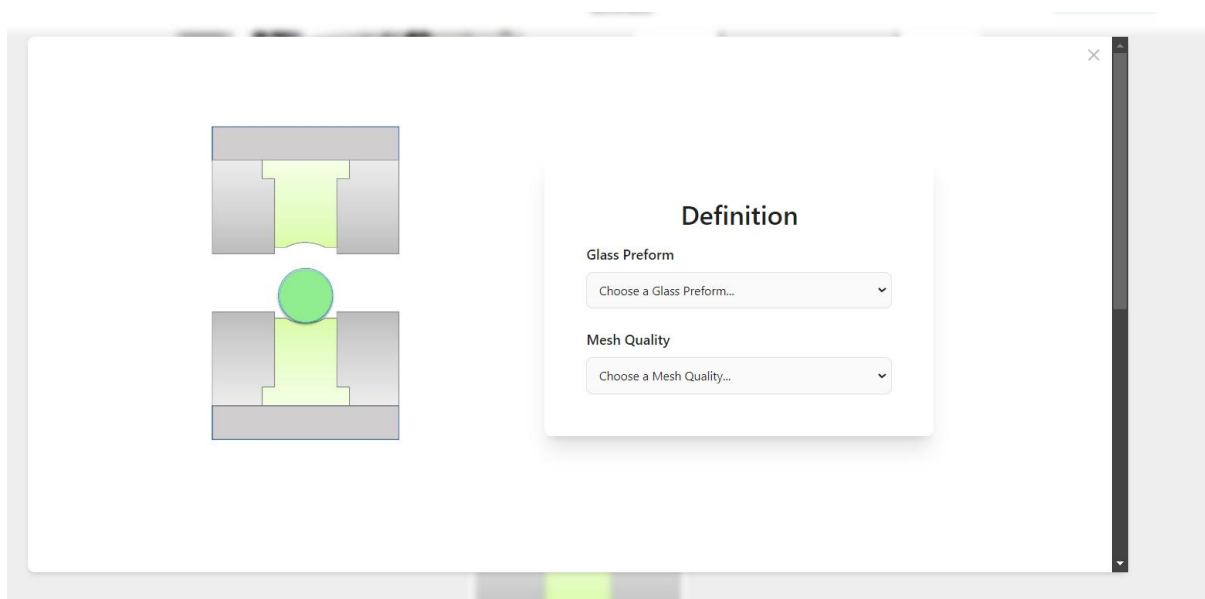


Figure 46 – Image Mapper with Glass Preform option selected <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

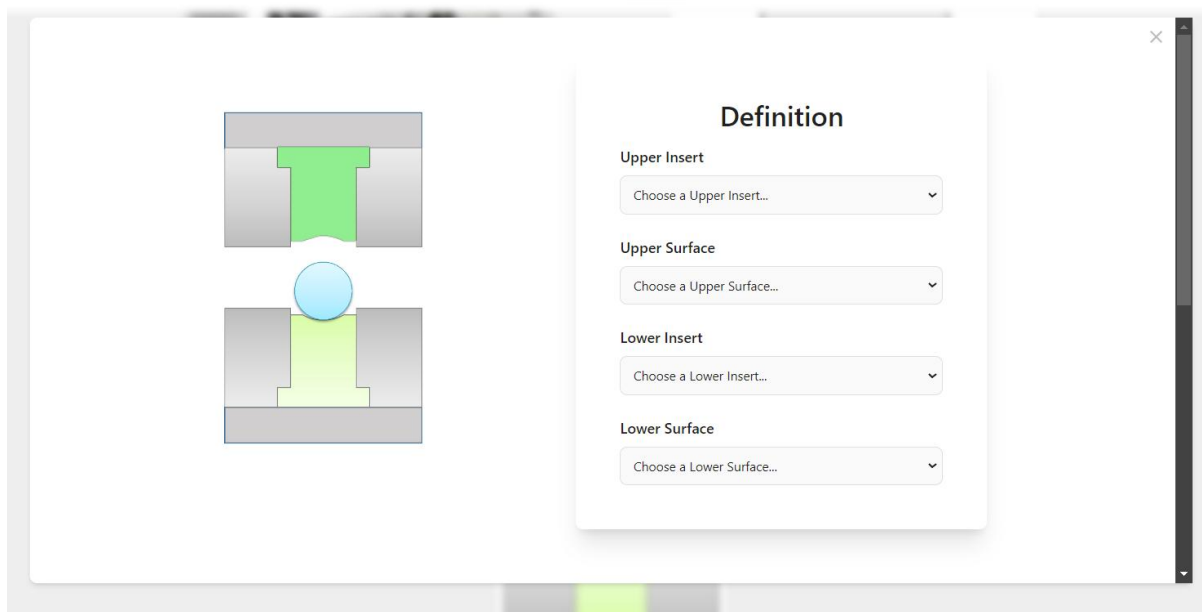


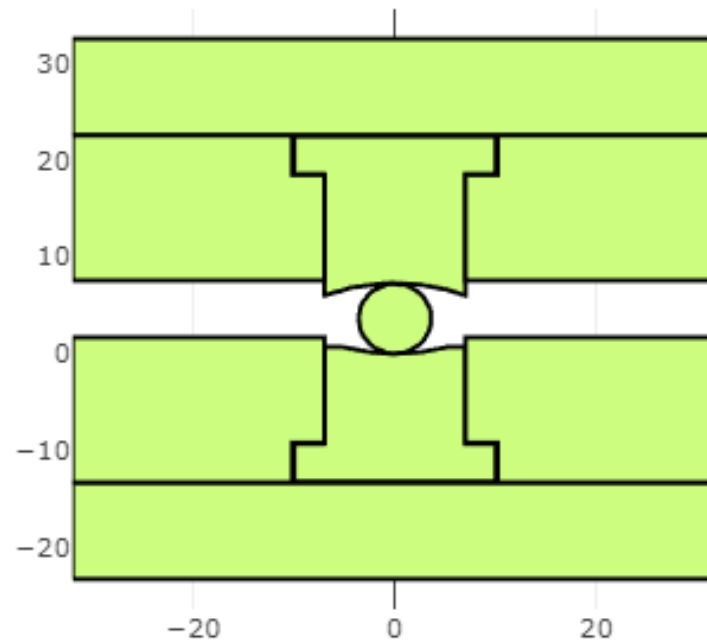
Figure 47 – Image Mapper with Insert option selected<sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

#### 5.4.5 Preview Molding Task Plot

The Preview Molding Task Plot displays all the elements involved in the molding process, offering a clear and detailed overview. It includes visual depictions of the tools and specific configurations chosen by the user, ensuring that they can see the complete setup before proceeding. This feature helps verify the correctness of the inputs and configurations, thereby reducing errors and enhancing the precision of the molding process.

The intuitive design of the Preview Molding Task Plot allows users to easily understand and modify their molding tasks. By providing immediate visual feedback, it supports better decision-making and streamlines the workflow, making the entire process more efficient and user-friendly, represented in the Figure 48

Figure 48 – Molding Task Plot <sup>1</sup>

<sup>1</sup> Picture resources in the figure belong to Fraunhofer IPT.

After filling in the requisite data within the modal, definition form, and cards, users can proceed by clicking on the "Create Molding Task" button. This action triggers the transmission of the accumulated data to the backend system, where the molding task is promptly generated and stored within the database. This newly created task is now primed for further processing, including transmission to ABAQUS for simulation generation. With a simple click, users can initiate the creation of their molding tasks, streamlining the workflow and expediting the simulation process.

## 6 CONCLUSION

### 6.1 CONCLUSIVE SUMMARY

This document presents the development and implementation of a web-based application module designed for conducting Finite Element Method (FEM) simulations, specifically focusing on the database generator and Molding task module. Through the integration of modern web technologies, including interactive user interfaces built with React and Tailwind CSS, and the implementation of advanced software architectures, we have created a tool that significantly streamlines the process of setting up and preparing data for FEM simulations.

The multi-tenant architecture of the SIMPGM project posed unique challenges, particularly in ensuring efficient and secure data retrieval. By developing a dedicated service to handle batch data operations, we successfully mitigated these performance issues, resulting in a more responsive and user-friendly application.

Key features of the module include the development of an intuitive and user-friendly interface, facilitating the selection and configuration of tools required for molding tasks. The tools preview plot provides visual feedback on the configured tools, enhancing accuracy and usability. Modal features were implemented for seamless user interactions and data inputs, while the chamfer table allows detailed and validated configuration of chamfers with automatic calculations and conditional selections. The Preview Molding Task Plot offers a comprehensive visualization of the entire molding process, ensuring users can see the complete setup and make necessary adjustments. Robust error handling mechanisms are in place to ensure data integrity and provide user guidance throughout the process.

These features collectively enhance the usability and functionality of the application, ensuring that users can efficiently configure their simulations, verify their setups, and make informed decisions based on visual and analytical feedback. In conclusion, the developed module not only improves the accessibility and usability of FEM simulation tools for engineers and researchers but also demonstrates the potential of modern web technologies in creating sophisticated and efficient software solutions.

### 6.2 FUTURE WORK

While this project has focused on the development and implementation of the database generator module for conducting FEM simulations, several additional enhancements and expansions are envisioned for the future to further augment the capabilities of the SIMPGM project.

A key area for enhancement is the integration of the simulation generator with the ABAQUS software. ABAQUS is a powerful tool for performing detailed and com-



plex FEM analyses, and integrating it with the SIMPGM software will streamline the process of running simulations. This integration will allow users to seamlessly transfer their configured simulation data from the SIMPGM interface to ABAQUS, execute the simulations, and retrieve the results directly within the SIMPGM environment. This workflow will not only save time but also reduce the potential for errors during data transfer, ensuring a more efficient and reliable simulation process. Another critical for future work involves the development of the Simulation Visualizer Module. This module will provide users with a comprehensive visual representation of the simulation results, offering advanced graphical tools to analyze and interpret the data. By incorporating features such as 3D visualization, interactive plotting, and result filtering, the Simulation Visualizer Module will significantly enhance the user's ability to understand and optimize their FEM simulations.

Furthermore, expanding the functionality of the simulation generator itself is essential. This includes automating more aspects of the simulation setup, providing advanced options for mesh generation, material properties specification, and boundary condition definition. By offering a more comprehensive set of tools within the simulation generator, users will have greater flexibility and control over their simulation setups.

In summary, the future development of the SIMPGM project will focus on enhancing visualization capabilities through the Simulation Visualizer Module, integrating seamlessly with ABAQUS for efficient simulation execution, and expanding the functionalities of the simulation generator. These advancements will collectively improve the overall user experience, making FEM simulations more accessible, efficient, and powerful for engineers and researchers.

## BIBLIOGRAPHY

- [1] MENZ, W. (Wolfgang), DIMOV, S. S., FILLON, Bertrand. In: **4M 2006 - Second International Conference on Multi-Material Micro Manufacture**. *Pages 251-254*; 2006.
- [2] WANG Li-li. **Foundations of Stress Waves, Numerical Methods for Stress Wave Propagation**. *Chapter 12, page 500*; 2007
- [3] ANH TUAN VU, DR.-ING. TIM GRUNWALD, PROF. DR.-ING. THOMAS BERGS. **Modeling, Lecture 9, High Precision Glass Optics Manufacturing**. *RWTH University*.
- [4] MARIO ANDRÉS PAREDES-VALVERDE, GINER ALOR-HERNÁNDEZ, ALEJANDRO RODRÍGUEZ-GONZÁLEZ, GANDHI HERNÁNDEZ-CHAN. **Developing Social Networks Mashups: An Overview of REST-Based APIs**. *2012 Iberoamerican Conference on Electronics Engineering and Computer Science*; 2012
- [5] BADR EDDINE SABIR, MOHAMED YOUSSEFI, OMAR BOUATTANE, HAKIM ALLALI. **Authentication and load balancing scheme based on JSON Token For Multi-Agent Systems**. *Settat University Hassan I, Mohammedia University Hassan II of Casablanca*; 2019
- [6] GOZDE KARATAS BAYDOGMUS, FERIT CAN, GAMZE DOGAN, CEMILE KONCA. **Multi-tenant architectures in the cloud: A systematic mapping study**. *Settat University Hassan I, Mohammedia University Hassan II of Casablanca*; 2019
- [7] MOCHAMMAD FARIZ SYAH LAZUARDY AND DYAH ANGGRAINI. **Modern Front End Web Architectures with React.Js and Next.Js.** *International Research Journal of Advanced Engineering and Science*; 2022
- [8] G. KARATAŞ, F. CAN, G. DOĞAN, C. KONCA AND A. AKBULUT. **Multi-tenant architectures in the cloud: A systematic mapping study**. *International Artificial Intelligence and Data Processing Symposium (IDAP)*; 2017
- [9] **Delivering MongoDB-as-a-Service: Top 10 Considerations**. MONGODB ORGANIZATION. **Delivering MongoDB-as-a-Service: Top 10 Considerations, Accessed on 09/01/2024**. *MongoDB Whitepaper*. Available at: <https://www.mongodb.com>.

- com/resources/products/platform/mongodb-service-top-10-considerations; 2021
- [10] SEBASTIÁN RAMÍREZ. **FastAPI Documentation, Accessed on 09/01/2024.** Available at: <https://fastapi.tiangolo.com/learn/>
- [11] SEBASTIÁN RAMÍREZ. **Fastapi JWT Auth Documentation, Refresh Tokens, Accessed on 09/01/2024.** Available at: <https://indominusbyte.github.io/fastapi-jwt-auth/usage/refresh/>
- [12] SDAMIAN PIWOWARCZYK. **Node.js MongoDB - multi-tenant app.** Available at: <https://dev.to/przpiw/nodejs-mongodb-multi-tenant-app-by-example-435n>; 2021
- [13] CODE SALLEY. **Multi-Tenancy with Nodejs and MongoDB, Accessed on 09/01/2024.** Available at: <https://dev.to/przpiw/nodejs-mongodb-multi-tenant-app-by-example-435n>; 2021
- [14] ROMAN RIGHT. **Beanie Documentation, Accessed on 09/01/2024.** Available at: <https://beanie-odm.dev/>
- [15] IPT, FRAUNHOFER. **Facts and Figures Fraunhofer IPT, Accessed on 09/01/2024.** Available at: <https://www.fraunhofer.de/en/about-fraunhofer.html>
- [16] MongoDB Organization. **MongoDB documentation, Accessed on 09/01/2024.** Available at: <https://www.mongodb.com/docs/drivers/python-drivers/>
- [17] AUTODESK. **Finite Element Analysis Solutions, Accessed on 11/05/2024.** Available at: <https://www.autodesk.com/solutions/simulation/finite-element-analysis>
- [18] ANSYS. **ANSYS Mechanical, Accessed on 11/05/2024.** Available at: <https://www.ansys.com/products/structures/ansys-mechanical>
- [19] Dassault Systèmes. **SIMULIA Abaqus, Accessed on 11/05/2024.** Available at: <https://www.3ds.com/products/simulia/abaqus>

- 
- [20] Red Hat. *What is a REST API?*, Accessed on 11/05/2024. Available at: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

# Appendix

## APPENDIX A – APPENDIX WITH CLASSIFIED DATA

This appendix contains code, data structures, and other materials that provide a clearer explanation of the implemented solution.

### A.1 CORE ./ - GET\_CURRENT\_USER FUNCTION

```

1 async def get_current_user(token: str = Depends(reusable_token)) ->
  TenantModel:
2     try:
3         # Decode the JWT token
4         payload = jwt.decode(
5             token,
6             settings.JWT_SECRET_KEY,
7             settings.ALGORITHM
8         )
9
10        # Extract token data
11        token_data = TokenPayload(**payload)
12
13        # Check token expiration
14        if datetime.fromtimestamp(token_data.exp) < datetime.now():
15            raise HTTPException(
16                status_code= status.HTTP_401_UNAUTHORIZED,
17                detail= "Expired token",
18                headers = {'WWW-Authenticate' : 'Bearer'}
19            )
20    except(jwt.JWTError, ValidationError):
21        # Handle token decoding/validation errors
22        raise HTTPException(
23            status_code = status.HTTP_403_FORBIDDEN,
24            detail = "Error in token authorization",
25            headers = {'WWW-Authenticate' : 'Bearer'}
26        )
27
28    # Connect to the TenantDB
29    await TenantService.connect_TenantDB()
30
31    # Retrieve tenant information based on username
32    tenant = await TenantService.get_tenant_by_username(token_data.sub
33    )
34
35    # Change database connection if user is not an admin
36    if token_data.sub != "admin":
37        await TenantService.change_db_connection(token_data.sub)

```

```
38     # Check if tenant exists
39     if not tenant:
40         raise HTTPException(
41             status_code=status.HTTP_404_NOT_FOUND,
42             detail="Tenant not found",
43             headers = {'WWW-Authenticate' : 'Bearer'}
44         )
45
46     # Return tenant information
47     return tenant
```

Listing A.1 – get\_current\_user function

## A.2 HELPERS ./GLASS\_PREFORM\_COORDS EXAMPLE

```
1 def glass_preform_coords(data):
2     new_data = [{
3         "Preform": {
4             "Type": data.preform.Type,
5             "Form Set": data.preform.form_set
6         }
7     }]
8     # Methods from the algorithms
9     assembly = Assembly(new_data)
10    coords = assembly.dynamic_drawing()
11
12    return coords
```

Listing A.2 – Python Fuction to generate the glass preform coordinates

## A.3 MIDDLEWARE ./ - CORS.PY FILE

```
1
2 from fastapi.middleware.cors import CORSMiddleware
3 from ..main import app
4
5
6 CORS_ORIGINS = [s
7     "http://localhost:3000", # Development server usign react
8     "http://localhost:5173", # Development server usign vite
9 ]
10
11 app.add_middleware(
12     allow_origins = CORS_ORIGINS, # List of origins that are allowed to
13     make requests to your FastAPI application
14     allow_credentials = True, # Determines whether credentials should be
15     included in cross-origin requests
```

```
14     allow_methods = ["*"], # Specifies the HTTP methods allowed
15     allow_headers = ["*"] # Specifies the HTTP headers that are allowed
16     in cross-origin requests.
17 )
```

Listing A.3 – CORS.py file

#### A.4 MODELS ./ - PROJECT MODEL

```
1 from beanie import Document
2 from .warehouse import Warehouse
3 from .IoTID import IoTID
4 from pymongo import IndexModel, DESCENDING
5 from typing import Type
6 from typing_extensions import Self
7 import datetime
8 class Project(Document):
9     # Logic Classified
10
11     @classmethod
12     async def max_nr(self):
13         """Find the number of class created elements
14
15         Returns:
16             int: number of elements created
17         """
18         # Logic Classified
19
20     @classmethod
21     def calculate_iot_id(cls: Type[Self], object: dict):
22         """Generates the document IOT-ID
23
24         Args:
25             cls (Type[Self]): a reference to the class itself. The Type[
26             Self] annotation is a type hint that specifies the class on which
27             this method is defined.
28             object (dict): Bson dictionary which the iot_id will be
29             generated.
30
31         Returns:
32             IoTID: Document IoTID
33         """
34         # Logic Classified
35
36     @classmethod
37     def create_label(cls: Type[Self], object: dict):
38         """Generate a label for the document
39
40         Args:
```



```
36     cls (Type[Self]): a reference to the class itself. The Type[
37     Self] annotation is a type hint that specifies the class on which
38     this method is defined.
39     object (dict): Bson dictionary which the iot_id will be
40     generated.
41
42     Returns:
43     str: document Label
44     """
45     # Logic Classified
46
47     class Settings:
48     # Document settings
49     name = 'Project' # Name of the document
50     union_doc = Warehouse # Union document for polymorphic behavior
51     indexes = [
52     IndexModel(
53     [("iot_id", DESCENDING)],
54     name="unique_iot_id",
55     unique=True, # Ensure uniqueness for iot_id field
56     ),
57     ]
```

Listing A.4 – Example of modal project file

## A.5 ROUTERS ./ - PROJECT ENDPOINTS

```
1
2 from fastapi import APIRouter, Depends, status, HTTPException
3 from typing import List
4 from core.JWTauth import get_current_user
5 from schemas.project_schema import ProjectSchema, ProjectCreateSchema
6 from models.tenant import TenantModel
7 from models.IoTID import IoTID
8 from models.project import Project
9 from services.project_service import ProjectService
10
11 project_router = APIRouter(
12     prefix='/project',
13     tags=['Database Generator', 'Project']
14 )
15
16 @project_router.get('/get-project', summary='Get a Project',
17     response_model=ProjectSchema, status_code=status.HTTP_200_OK)
18 async def get_project(iot_id: IoTID, current_user: TenantModel = Depends
19     (get_current_user)) -> ProjectSchema:
20     project = await ProjectService.get_project(iot_id=iot_id)
```

```
19     if not project:
20         raise HTTPException(
21             status_code=status.HTTP_404_NOT_FOUND,
22             detail="The project you are trying to query doesn't exist."
23         )
24     return project
25
26 @project_router.get('/get-projects', summary='Get a Project List',
27     response_model=List[ProjectSchema], status_code=status.HTTP_200_OK)
28 async def get_projects(current_user: TenantModel = Depends(
29     get_current_user)) -> List[ProjectSchema]:
30     projects = await ProjectService.get_projects()
31     if not projects:
32         raise HTTPException(
33             status_code=status.HTTP_404_NOT_FOUND,
34             detail="No projects found."
35         )
36     return projects
37
38 @project_router.post('/create-project', summary='Create a Project',
39     response_model=ProjectSchema, status_code=status.HTTP_201_CREATED)
40 async def create_project(data: ProjectCreateSchema, current_user:
41     TenantModel = Depends(get_current_user)) -> ProjectSchema:
42     try:
43         return await ProjectService.create_project(data=data)
44     except Exception as error:
45         raise HTTPException(
46             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
47             detail=f"An error occurred while creating the project: {str(
48                 error)}")
```

Listing A.5 – Example of Project router file

## A.6 SERVICES ./ - PROJECT SERVICES

```
1 from models.tenant import TenantModel
2 from schemas.project_schema import ProjectSchema, ProjectCreateSchema
3 from models.project import Project
4 from models.IoTID import IoTID
5 from typing import List
6
7 class ProjectService:
8
9     @staticmethod
10     async def create_project(data: ProjectCreateSchema) -> Project:
11         max_nr = int(await Project.max_nr()) + 1
```

```
12     project = data.dict()
13     project['nr'] = max_nr
14     project['iot_id'] = Project.calculate_iot_id(project)
15
16     if project['label'] == None or project['label'] == "":
17         project['label'] = Project.create_label(project)
18
19     new_project = Project(
20         **project
21     )
22     return await new_project.insert()
23
24     @staticmethod
25     async def get_project(iot_id: IoTID) -> ProjectSchema:
26         target_project = await Project.find_one(Project.iot_id == iot_id
27     )
28
29         return target_project
30
31     @staticmethod
32     async def get_projects() -> List:
33         projects = await Project.find().to_list()
34         return projects
```

Listing A.6 – Example of Project service file

```
1 import axiosInstance from "../APISettings"
2 // POST endpoint
3 export const createProject = (data) => {
4     const endpointURL = "/project/create-project"
5     const new_project = {
6         "project_name": data.projectName,
7         "description": data.description,
8         "label": ""
9     }
10
11     return axiosInstance.post(
12         endpointURL,
13         new_project
14     )
15 }
16
17 // GET endpoint
18 export const getProjects = async () => {
19     const endpointURL = "/project/get-projects"
20     return axiosInstance.get(
21         endpointURL,
22     )
```

23

## Listing A.7 – Project API endpoints

## A.7 CONTEXTS - MODAL CONTEXT EXAMPLE

```
1 import { createContext } from "react";
2 import { useState } from "react";
3 export const ModalContext = createContext()
4
5 export const ModalProvider = ({ children }) => {
6
7   const [modalArray, setModalArray] = useState({
8     // Data Structure Classified
9   })
10
11   const [rowsS1, setRowsS1] = useState([]);
12   const [rowsS2, setRowsS2] = useState([]);
13   const [rowsInsert, setRowsInsert] = useState([]);
14
15   const [insertArray, setInsertArray] = useState({
16     // Data Structure Classified
17   })
18
19   const [definitionFormArray, setDefinitionFormArray] = useState({ "":
20     "" });
21   const [showNoneOD, setShowNoneOD] = useState(false)
22   const [showNoneIN, setShowNoneIN] = useState(false)
23   const [flatIMG, setFlatIMG] = useState(null)
24   const hideFlat = () => setFlatIMG(false)
25   const showFlat = () => setFlatIMG(true)
26
27   return (
28     <ModalContext.Provider value={
29       {
30         modalArray, setModalArray, rowsS1, setRowsS1, rowsS2,
31         setRowsS2, insertArray, setInsertArray, definitionFormArray,
32         setDefinitionFormArray, rowsInsert, setRowsInsert, showNoneOD,
33         setShowNoneOD, showNoneIN, setShowNoneIN, hideFlat, showFlat,
34         flatIMG
35       }
36     }
37     >
38       {children}
39     </ModalContext.Provider>
40   )
41 }
```

36

Listing A.8 – Context code example with the modal Context for Optical Design and Insert

## A.8 GETTING USER DATA - DATA CAPTURE EXAMPLE

```
1
2 import { useForm } from 'react-hook-form'
3
4 // Store input information and handle the form submission.
5 const { register, handleSubmit } = useForm();
6
7 const onSubmit = (data) => {
8     setShowAlert(false)
9     createProject(data).then(
10         response => {
11             console.log(response.data)
12             setShowAlert(true)
13             // Set the message and a Alert for successful form
14             submiton
15             setAlertMessage('Project ${data.projectName} Successfully
16             Created!')
17             setAlertColor("green")
18             // Redirects the user to the next generator
19             setTimeout(() => {
20                 navigate("/database-generator/optical-design");
21             }, 1100);
22         }
23     ).catch(
24         (error) => {
25             console.log(error)
26             setShowAlert(true)
27             // Set the message and a Alert for form submiton with
28             error and displays the errors defined in the backend API
29             setAlertColor("red")
30             setAlertMessage(error.response.data.detail)
31         }
32     )
33 }
34
35 return (
36     <>
37
```

```
38 <form onSubmit={handleSubmit(onSubmit)} className="mx-auto mt-32 md:mt
    -48 lg:mt-48">
39   // ... Parent components
40
41   <FormTextInput
42     label={'Project Name'}
43     placeholder={'Project Name'}
44     formEvent={{
45       // The Register function is declared here
46       ...register("Code Classified", {
47
48         required: true,
49       })
50     }}
51     defaultValue = {inputValuesMapper['Code Classified'] &&
inputValuesMapper['Code Classified']}
52   />
53
54   // ... Other Child Components
55   <SubmitFormButton
56     size="full"
57   >
58     Create a Project
59   </SubmitFormButton>
60 </form>
```

Listing A.9 – User data capture example

## A.9 PDF VISUALIZER CODE IMPLEMENTATION

```
1
2 export async function pdfToString(file) {
3   return new Promise((resolve, reject) => {
4     const reader = new FileReader();
5
6     // Event handler for successful file read
7     reader.onload = function (event) {
8       const pdfData = event.target.result; // The base64-encoded
data URL
9       resolve(pdfData); // Resolve the promise with the data URL
10    };
11
12    // Event handler for file read error
13    reader.onerror = function (error) {
14      reject(error); // Reject the promise with the error
15    };
16  }
```

```

17     reader.readAsDataURL(file); // Read the file as a data URL (
    base64-encoded string)
18   });
19 }
20
21 export async function stringToPdf(pdfData) {
22   // Extract the base64-encoded string (without the data URL prefix)
23   const base64Data = pdfData.split(',')[1];
24
25   // Decode the base64-encoded string to a byte string
26   const byteCharacters = atob(base64Data);
27
28   // Create an array of byte numbers
29   const byteNumbers = new Array(byteCharacters.length);
30   for (let i = 0; i < byteCharacters.length; i++) {
31     byteNumbers[i] = byteCharacters.charCodeAt(i);
32   }
33
34   // Convert the byte numbers array to a Uint8Array
35   const byteArray = new Uint8Array(byteNumbers);
36
37   // Create a Blob object from the Uint8Array with the MIME type '
application/pdf'
38   const blob = new Blob([byteArray], { type: 'application/pdf' });
39   return blob; // Return the Blob object
40 }

```

Listing A.10 – Binary conversions

```

1 export const PDFVisualizer = ({ drawingSrc, className }) => {
2   return (
3     <div className={className}>
4       <embed src={drawingSrc}
5         className={['border-4 lg:w-screen md:w-screen w-11/12 h-[400px]
6         mt-1 md:h-[600px] lg:h-[600px] rounded-md border-gray-900']>
7       </embed>
8     </div>
9   );
10 export default PDFVisualizer;

```

Listing A.11 – PDFVisualizer component

Another requisite of the PDFVisualizer will need the FileInputComponent from the project UI and the handlePDF function.

```

1 <FormFileInput
2   formEvent={{
3     ...register("drawing", {
4       onChange: (event) => {

```

```

5         handlePDF(event.target.files[0])
6     }
7     })
8 }}
9 />

```

Listing A.12 – formFile Input Component

The handlePDF function is an asynchronous function designed to process a PDF file and update the component's state with both a URL for displaying the PDF and its binary string representation.

```

1
2 const handlePDF = async (pdfFile) => {
3     // Convert the PDF file to a binary string representation using the
4     pdfToString function. The await keyword ensures this asynchronous
5     operation completes before moving to the next line.
6     const binaryPDF = await pdfToString(pdfFile);
7
8     // Create an object URL from the PDF file to be used for displaying
9     the PDF in the UI. The URL.createObjectURL method creates a string
10    containing a URL representing a object.
11    const newPDF = URL.createObjectURL(pdfFile)
12
13    // setDisplayedPDF is a state setter
14    setDisplayedPDF(newPDF);
15
16    // Update the component's state with the binary string
17    representation of the PDF.
18    setDrawingPDF(binaryPDF);
19 };

```

Listing A.13 – handlePDF function

## A.10 DROP-DOWN DATA IMPLEMENTATION

```

1 @staticmethod
2     async def get_projects() -> List:
3     # Get All Documents from the Project collection and turn in a list
4     projects = await Project.find().to_list()
5     return projects

```

Listing A.14 – Isolated Services from project - Previous Approach

```

1 @staticmethod
2     async def get_tool_materials() -> List:
3     # Get All Documents from the ToolMaterial collection and turn in a
4     list

```



```

4     tool_materials = await ToolMaterial.find().to_list()
5     return tool_materials

```

Listing A.15 – Isolated Services from toolMaterial

```

1 @staticmethod
2     async def get_sleeve_dropdown_data() -> SleeveDropdownSchema:
3         tool_materials = await ToolMaterialService.get_tool_materials()
4         project = await ProjectService.get_projects()
5
6         return SleeveDropdownSchema(
7             projects= project,
8             tool_materials= tool_materials
9         )

```

Listing A.16 – Dropdown services

```

1 class SleeveDropdownSchema(BaseModel):
2     projects: List
3     tool_materials: List

```

Listing A.17 – Sleeve Data Schema

## A.11 PREVIEW PLOT IMPLEMENTATION

Frontend Endpoint send the data request:

```

1 export const createGlassPreformCoordinates = (data) => {
2     const endpointURL = "/glass-preform/create-glass-preform-plot"
3     return axiosInstance.post(
4         endpointURL,
5         data)
6 }

```

Listing A.18 – Glass Preform Plot Frontend endpoint

API endpoint responsible to calculate the coord and get the coord back.

```

1 @glass_preform_router.post("/create-glass-preform-plot", summary= "
2     Create coordinates to plot Optical Design in preview plot component."
3     , status_code=status.HTTP_200_OK)
4 async def create_glass_preform_coordinates(data:GlassPreformPlotSchema):
5     coords = glass_preform_coords(data)
6     return coords

```

Listing A.19 – Glass Preform Plot Backend endpoint

Function usign the algorithm methods to calculate the coords

```

1 def generate_concept_coords(data):
2     assembly = Assembly(data)
3     coords = assembly.dynamic_drawing()

```

```
4 return coords
```

### Listing A.20 – Function using the algorithm methods

Plot Component:

```
1 import Plot from "react-plotly.js";
2 import Spinner from "../../UI/Spinner/Spinner";
3
4
5 export const PreviewPloty = (props) => {
6
7   if (props.loading) return (
8     <div className="flex items-center justify-center w-full pt-40 pb
9     -60">
10       <Spinner />
11     </div>)
12   return (
13     <Plot
14       className={props.className}
15       data={[
16         {
17           x: props.data && props.data[0] && props.data[0].x,
18           y: props.data && props.data[0] && props.data[0].y,
19           type: "scatter",
20           mode: "lines",
21           marker: { color: "black" },
22           fill: props.fillPlot ? "toself" : "",
23           fillcolor: "#bbefff",
24           name: props.name,
25         },
26       ]}
27       layout={{
28         yaxis: {
29           scaleanchor: "x",
30           scaleratio: 1,
31         },
32         xaxis: {
33           layer: "above traces",
34           // Dynamically set the x-axis range to include one
35           // additional value
36           range: props.data && props.data[0] && props.data[0].
37           x ? [
38             Math.min(...props.data[0].x) - 1 ,
39             Math.max(...props.data[0].x) + 1, // Assuming
40             the step between x values is 1
41           ] : undefined,
42         },
43       }}
44     >
```

```
40     />  
41   })  
42   export default PreviewPloty;
```

Listing A.21 – Plot Component