FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Web Application for Cost and Manufacturing Event Management in Formula Student Competitions
Student:	Viktória Ritzková
Supervisor:	Ing. Marek Suchánek, Ph.D. et Ph.D.
Study program:	Informatics
Branch / specialization:	Software Engineering 2021
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2024/2025

Instructions

Formula Student is the largest technology student competition in the world. The competition is held every year across the world and part of it is a cost-related discipline that requires a lot of precise documentation. The goal of this thesis is to prepare in collaboration with eForce Prague Formula team (FEL & FME CTU) a web application that efficiently supports collection of the relevant information and production of the required documentation. The application shall be developed following the software engineering methods:

- Describe the Formula Student competition and focus on the Cost & Manufacturing event. Analyze what information is needed, what and how is calculated or inter-related.
- Analyze how the eForce team is currently dealing with the Cost & Manufacturing (what tools are used, how information is collected and captured etc.) and identify potential optimizations.
- Set functional and non-functional requirements on the new web application. Prepare use cases related to the functional requirements.
- Design the web application fulfilling the requirements using .NET technologies. Take into account maintainability, sustainability, and extensibility of the application.
- Implement a prototype of the application based on the design. Justify use of additional technologies, document and test the prototype.
- Evaluate the results of your work and outline possible future development.

Electronically approved by Ing. Michal Valenta, Ph.D. on 31 October 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Web Application for Cost and Manufacturing Event Management in Formula Student Competitions

Viktória Ritzková

Department of Software Engineering
Supervisor: Ing. Marek Suchánek, Ph.D. et Ph.D.

May 16, 2024

Acknowledgements

I would like to thank my supervisor, Marek Suchánek, Ph.D. et Ph.D., for his professional guidance on my bachelor's thesis and for the advice provided during the course of the development of this thesis. I want to thank team eForce Prague Formula for the opportunity to participate in the Formula Student project and for their cooperation in the implementation of my bachelor's thesis. Last but not least, I want to thank my family and friends who supported me throughout my studies and thanks to whom I was able to make it this far.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 16, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Ritzková, Viktória. *Web Application for Cost and Manufacturing Event Management in Formula Student Competitions.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstrakt

Tato práce se zabývá vývojem webové aplikace ve spolupráci s týmem Force Prague Formula, která slouží ke sběru dat v disciplíně Cost & Manufacturing soutěže Formula Student. V souladu se standardními postupy softwarového inženýrství začíná práce analýzou disciplíny s cílem definovat požadavky pro vyvíjenou aplikaci. Následně je aplikace navržena, implementována a testována. V závěru práce jsou vyhodnoceny výsledky a nastíněny možnosti dalšího vývoje. Výsledkem práce je funkční aplikace připravená k produkčnímu nasazení.

Klíčová slova Webová aplikace, Formula Student, Cost & Manufacturing, Blazor, .NET

Abstract

This thesis is dedicated to the development of a web application, in collaboration with the eForce Prague Formula team, used for data collection in the Cost & Manufacturing discipline of the Formula Student competitions. Following standard software engineering practices, the thesis begins with the event's analysis to define application's requirements. Subsequently, the application is designed, implemented, and tested. The thesis concludes with an evaluation of results and outlines possibilities for future development. The result is a functional application ready for production deployment.

Keywords Web Application, Formula Student, Cost & Manufacturing, Blazor, .NET

Contents

List of Figures

List of Tables

Introduction

At the beginning of the 1980s, a few passionate students from Texas, USA decided to build a custom racing single-seater, and thus Formula Student was born. Since then, the competition spread across the globe and year by year new aspiring engineers step into the world of motorsport and push the boundaries of technology and innovation forward. Through this opportunity, they gain experience in technical design, hands-on manufacturing, and, nonetheless, teamwork.

In today's business world, effective cost management and innovative production processes are more essential for success than ever. The Cost & Manufacturing discipline of the competition therefore aims to evaluate the team's abilities to finance race car production, their strategies for said cost management, as well as their comprehensive understanding of manufacturing processes. These are all essential traits for successful business ownership. Financing Formula Student teams can be particularly challenging, as they often operate with limited financial and/or personnel resources.

The main goal of this thesis is to prepare in collaboration with eForce Prague Formula a web application that supports the collection of relevant information necessary for the Cost & Manufacturing discipline as well as the production of required documentation. The thesis further contains the analysis, design, and implementation of the application, following the standard principles of any software development.

The beginning of the thesis focuses on the Formula Student competition as a whole, describes its history and concept, presents a brief overview of all the disciplines in the competition, and introduces team eForce Prague Formula. Subsequently focus shifts to the Cost & Manufacturing discipline covering its rules and requirements and accesses the current approach of the team to the discipline, followed by the analysis of software requirements, as well as creation of use cases and the class diagram. The practical part of the tesis is dedicated to the possibilities of .Net development and the software design of the application. It records the factual implementation as well as testing, which is a crucial part of any software development. The thesis is concluded by assessment of the findings, the final implementation, and outlines the possibilities for future development.

Chapter 1

Goals

The main goal of this thesis is to prepare, in collaboration with eForce Prague Formula Team, a web application that efficiently supports the collection of the relevant information for the Cost & Manufacturing discipline in Formula Student Competition and the production of the required documentation.

To achieve the primary goal, several partial objectives must be fulfilled.

Firstly, it is necessary to describe the Formula Student competition and focus on the Cost & Manufacturing event. Analyze what information needs to be collected and how it should be categorized, as well as analyze the team's current approach to the discipline and identify potential optimizations.

The second objective is to design the web application fulfilling the requirements using .NET technologies, taking into account maintainability, sustainability, and extensibility of the application.

The analysis should then be followed by implementation of a prototype based on the design. The use of additional technologies should be justified.The prototype should then be properly documented and tested.

Finally it is necessary to evaluate the results of this thesis and outline possible future development.

Chapter 2

Formula Student Competition

Formula Student is an international engineering competition, in which groups of passionate students create formula-like vehicles with which they compete on various international circuits. Today, nearly 800 teams from around the world compete in European competitions, as well as in the United States, Brazil, Japan, India, and Australia. The aim of the competition is for students to establish contacts in the automotive industry, gain practical experience and project management skills, and, above all, design and manufacture safe, reliable, and efficient vehicles. In the races, not only the vehicle's driving abilities are tested, but also the overall development, budget, and sales potential. The competition consists of two categories: the Electrical vehicle (EV) category and the Driverless vehicle (DV) category. Historically, there was also a Combustion engine vehicle (CV) category, but since the 2024 season, most races have withdrawn from this division. [1]

2.1 History of Formula Student

In 1981, The Society of Automotive Engineers (SAE) in the USA started the Formula SAE competition, in which almost 140 teams competed in building one-seaters with combustion engines. Seventeen years later, SAE, together with the Institution of Mechanical Engineers (IMechE), brought the competition to Europe, and Formula Student was created. In 2006, the competition spread also to Australia, and Formula SAE Australasia was created. All the divisions exist to this day, with each having similar rules to one another, therefore enabling the teams to compete across the divisions. In 2010, the competition introduced the EV division, and in 2017, the DV division, in which the vehicles run fully autonomously. [2]

2.2 Disciplines

The Formula Student races in Europe follow the rules of Formula Student Germany. The teams compete in 3 static disciplines and up to 10 dynamic disciplines. Each discipline is rewarded by a certain number of points, as displayed in Figure 2.1. However, before the car is eligible to compete on the racetrack, it must complete various technical inspections, ensuring that

the car is rules-compliant and, more importantly, safe to drive. A more in-depth overview of the individual disciplines is presented below based on the competition rules [3].

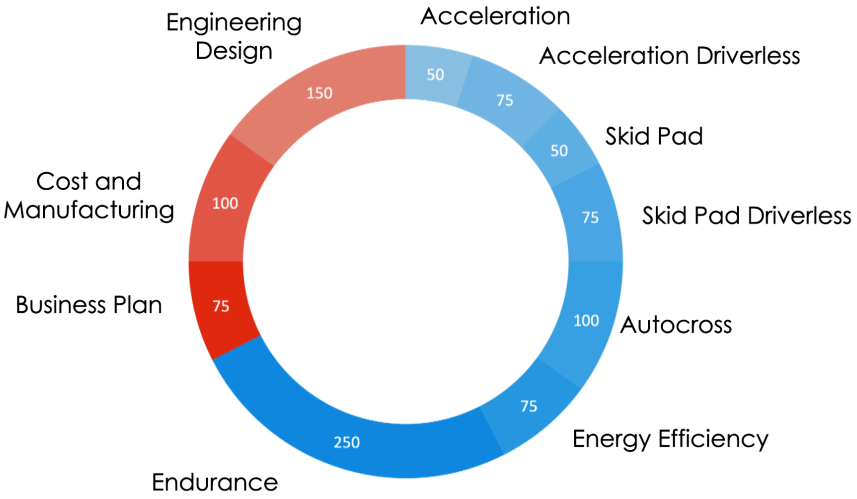


Figure 2.1: Points distribution. [4]

2.2.1 Statics

Formula Student tries to introduce students to all aspects of the automotive industry. Therefore, static events test future engineers on their presentation techniques and financial planning skills.

2.2.1.1 Engineering Design

Prior to the start of this event, the team is required to submit an 8-page technical description of the vehicle. The document should include the design and the manufacturing process. Based on the document, the panel of judges will evaluate the vehicle's technical design, construction, and overall production. The evaluation is then followed by a discussion between the judges and the students to better evaluate the technical specification and the overall thinking and reasoning behind the chosen design. The score then reflects not only the quality of the design but also the decisions behind it.

2.2.1.2 Cost & Manufacturing

Cost is the deciding aspect of all products in today's economy. The same applies to the vehicles that are build by the teams. Therefore, students are required to hand in an extensive written report documenting all parts present in the vehicle and step-by-step manufacturing processes needed for their assembly.

2.2.1.3 Business Plan

Each team presents a fictional business plan for a product or service, which references the team's current vehicle. The aim is to create a lucrative business opportunity with the highest overall profit. Each team has 10 minutes to pitch their product to the judges posing as potential investors.

2.2.2 Dynamics

Even though the design and financial aspects of vehicle construction are crucial, the most points in the competitions come from dynamic events in which the car displays its performance, speed, and reliability, therefore putting the team's theoretical knowledge to the test. Each discipline focuses on and tests a different aspect of the vehicle.

2.2.2.1 Acceleration

The vehicle's acceleration is measured over a 75 m long track with a fixed start. The best teams are able to complete it in under 3 seconds, reaching over 120 km/h.

2.2.2.2 Skidpad

The skidpad track, built in the shape of a figure 8, tests the maximal lateral acceleration of the vehicle. Many aspects create the best run, such as good tires, good aero packet, low central of mass, and last but not least, a concentrated driver.

2.2.2.3 Autocross

Autocross is a 1km long track consisting of various straights, curves, slaloms, and chicanes. As a result, every little design aspect shows and ultimately saves or adds milliseconds to the resulting time. Additionally, the driver's abilities are crucial for achieving the best possible time: precise handling, on-time braking, and acceleration are all aspects of a winning run. This event is often referred to as qualifying because the final ranking determines the starting order for the endurance race.

2.2.2.4 Endurance

Providing the most points, this event is the central discipline of the competition, in which the cars prove their durability under long-term conditions. The track is almost identical to the autocross track, but instead of one lap, the car is driven for 22km with a driver change after 11th km.

2.2.2.5 Efficiency

During endurance, energy consumption is recorded. The final efficiency score is then calculated as the difference in energy between the start of the race and the end of the race relative to speed. This prevents the team from driving slowly in order to save energy and, therefore, score more points. Additionally, as the

vehicles run on electrical accumulators the aspect of recuperation[1] comes into play, as the recuperated energy is subtracted from the total used energy.

2.3 eForce Prague Formula

Student race cars have been built under the Faculty of Mechanical Engineering at CTU since 2007 by the CTU Cartech team. Over the 15 years, they successfully built 15 single-seater cars with combustion engines, the last two generations being hybrid.

The student team eForce FEE Prague Formula was established in 2010 when a few enthusiastic CarTech members decided to leave their previous team and start developing the first electric formula-like vehicle in the Czech Republic. [4]

In 2019, eForce once again inscribed itself into the history of Czech motorsport when it released the first fully autonomous electric formula built in the Czech Republic. The year 2022 was the team's most successful season, with eight victories, 6 second and 4 third places across various disciplines. That year, the team ranked at an incredible 17th place in the Formula Student Electric World Ranking [6].

The future of Formula Student lies in the autonomous electric vehicles; therefore, in 2024, both teams, eForce FEE Prague Formula and CTU Cartech, decided to join forces and establish a new team, eForce Prague Formula, based at the Faculty of Electrical Engineering and Faculty of Mechanical Engineering at CTU in Prague. Currently, the team has approximately 90 members from various CTU faculties with few members from other universities. [4]

[1] Recuperation is a process of recovering some of the kinetic energy lost in slowing down the car or braking. [5]

Chapter 3

Cost & Manufacturing event

The Cost & Manufacturing discipline aims to access the team's understanding of the manufacturing process and the financing associated with constructing a prototype. In this chapter, we will focus on the discipline procedure described in the rules [7] and the team's current approach to this event.

3.1 Procedure

Prior to the competition, the team has to submit three Cost Report Documents (CRD) to the competition's website. At the competition, a discussion with the judges will take place next to the team's vehicle.

The discussion consists of three parts:

- **Bill of Materials (BOM) discussion**

  The judges access the team's ability to create accurate engineering and manufacturing BOM for the vehicle. The BOM has to reflect the vehicle brought to the competition accurately, all costs and calculations included are accurate and realistic, and the BOM has to reflect the manufacturing feasibility of the vehicle.

- **Cost Understanding**

  Discussion about the team's overall knowledge of finances and manufacturing processes.

- **Real case**

  A few days prior to the competition, there will be a specific task about cost and manufacturing in a particular field posted on the competition's website, the goal of which is to put the team's knowledge to the test.

3.2 Cost Report Documents

The Cost Report Documents (CRD) consist of:

- the BOM including the Costed Bill of Materials (CBOM) and the Detailed Bill of Materials (DBOM),

- the supporting material file,
- the cost explanation file.

The documents have to be submitted to the competition's website by a specific deadline, and the team is required to bring at least one hard copy to the competition.

3.2.1 Bill of Materials

The Bill of Materials is a sorted list of all vehicle parts and must include all parts of the vehicle brought to the competition. The BOM is structured as follows:

- The BOM is divided into *systems*. *The systems* are predefined by the competition and unchangeable. The list of all systems can be viewed in Table A.1.
- Each *system* is divided into *assemblies*, which are also predefined by the competition as shown in Table A.2, but unlike the systems, can be further specified by the teams.
- Each *assembly* can be broken down into *subassemblies* which are specified by the team.
- Each *assembly* or *subassembly* is broken down into *parts* that are specified by the team.

Fasteners are additional items used to assemble a *part* and should not be included, provided they are not self-manufactured. All self-manufactured *fasteners* are considered part of the assembly.

All entries of the BOM should include:

- name, which must clearly describe the part,
- comment,
- and an optional custom ID. If left blank, the competition website will generate the ID.

The BOM must use metric units only.

3.2.2 Detailed Bill of Materials

In addition to the information included in the BOM, the Detailed Bill of Materials also includes the manufacturing and assembly processes of all parts from two or three systems included in the BOM. The additional information is as follows:

- Each part is broken down into:
  - *Material* – Meaning raw material of the part, e.g., aluminum.

  - *Processes* – Process is the operation necessary to produce the part from the material. Furthermore, if necessary, processes can be broken down into *fasteners* and *tooling*. *Tooling* represents the necessary tools used to change material into a specific shape. For example, welding jigs, molds, and others.

- For each *part* in the *assemblies*, it has to be specified whether it was bought or made. For each bought *part*, only the *fasteners* are included; however, if the *part* was altered from its original state, all *processes* of the change need to be included as well.

3.2.3 Costed Bill of Materials

The Costed Bill of Materials includes all information included in the BOM and the DBOM, and additionally, it includes the accurate cost of all *parts* from one or two DBOM systems. These systems are specified in the competition handbook in advance.

Cost calculations should include the costs of material, manufacturing, and purchased parts, as well as the cost of the assembly process. All these calculations should aim to be as close to reality as possible. All costs must be displayed in EUR and estimations of hourly work rates and machining should be included.

On the other hand, the calculations should exclude development, research, and capital expenses for real estate, such as the development hours of the team. The costs of power tools should also be excluded.

An snippet of the team's BOM submitted at the 2023 Formula Student Czech Republic can be viewed in Figure A.1.

3.2.4 Supporting Material File

The Supporting Material File contains additional information to better the comprehension of the BOM. It usually consists of drawings, exploded view drawings, and pictures of the parts or assemblies included in the vehicle.

3.2.5 Cost Explanation File

The Cost Explanation file explains the team's overall approach to the discipline. It includes all the necessary calculations, such as hourly rates in the Czech Republic, conversion between EUR and CZK used, and calculations of all manufacturing prices.

3.3 Scoring

The Cost and Manufacturing event will be evaluated as shown in the table 3.1. If items are missing in any of the documents, the points will be deducted from the "BOM and BOM Discussion" category, as shown in the table 3.2

Category	**Points**
Format and Accuracy of Documents	5
Knowledge of Documents and Vehicle	5
BOM and BOM Discussion	35
Discussion Part 2 "Cost Understanding"	35
Part 3 "Real Case"	20
Total	100

Table 3.1: Cost and Manufacturing Scoring, based on [7]

Missing item	**Points**
Assembly	-5
Part	-3
Process Material	-1

Table 3.2: Cost and Manufacturing score deductions, based on [7]

3.4 Current Approach Analysis

This section comperes the tool presented by the competition and an approach of team eForce Prague Formula to the discipline.

3.4.1 FSG Portal

The competition website provides the team with a tool for creating all BOMs named FSG Portal as seen in Figure 3.1. It requires the BOM to be structured, as per the rules, into *systems* and *assemblies* (possibly *subassemblies*), and also requires the *processes* and costs in the DBOM and CBOM of *systems*, which were specified by the competition handbook.

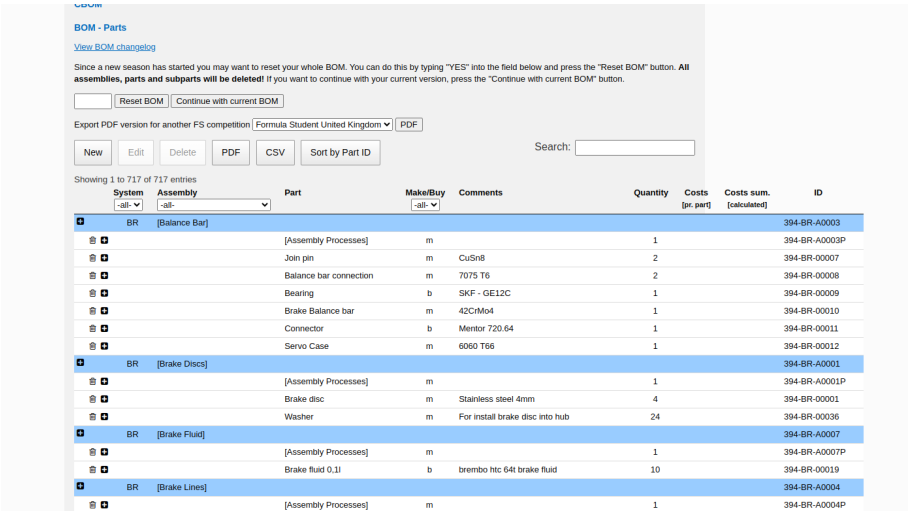
The portal has several useful features such as:

- ability to enter a custom ID,
- ability to export the data to a document,
- alerts the user if the costs are not present where they are required.

However, there are also several features that the system does not have, which sparked the idea of creating a new independent tool. The missing features from the team's point of view are:

- There is no undo button or soft delete, meaning all deleted items will be lost permanently.
- It does not support adding material, processes, and costs to the systems that do not require it for the said season. Team eForce would like to collect these data to improve their knowledge and understanding of the car manufacturing process.
- It does not allow to attach support files directly to the collected data.

The further visualisation of FSG portal can be viewed in Figures A.2 to A.5



BOM - Parts

View BOM changelog

Since a new season has started you may want to reset your whole BOM. You can do this by typing "YES" into the field below and press the "Reset BOM" button. **All assemblies, parts and subparts will be deleted!** If you want to continue with your current version, press the "Continue with current BOM" button.

Reset BOM | Continue with current BOM

Export PDF version for another FS competition: Formula Student United Kingdom | PDF

New | Edit | Delete | PDF | CSV | Sort by Part ID | Search:

Showing 1 to 717 of 717 entries

System	Assembly	Part	Make/Buy	Comments	Quantity	Costs [pr. part]	Costs sum. [calculated]	ID
BR	[Balance Bar]							394-BR-A0003
		[Assembly Processes]	m		1			394-BR-A0003P
		Join pin	m	CuSn8	2			394-BR-00007
		Balance bar connection	m	7075 T6	2			394-BR-00008
		Bearing	b	SKF - GE12C	1			394-BR-00009
		Brake Balance bar	m	42CrMo4	1			394-BR-00010
		Connector	b	Mentor 720.64	1			394-BR-00011
		Servo Case	m	6060 T66	1			394-BR-00012
BR	[Brake Discs]							394-BR-A0001
		[Assembly Processes]	m		1			394-BR-A0001P
		Brake disc	m	Stainless steel 4mm	4			394-BR-00001
		Washer	m	For install brake disc into hub	24			394-BR-00036
BR	[Brake Fluid]							394-BR-A0007
		[Assembly Processes]	m		1			394-BR-A0007P
		Brake fluid 0,1l	b	brembo htc 64t brake fluid	10			394-BR-00019
BR	[Brake Lines]							394-BR-A0004
		[Assembly Processes]	m		1			394-BR-A0004P

Figure 3.1: FSG Portal [8]

3.4.2 Team's Data Collection

Team eForce used the FSG Portal for the data collection, as there was no other viable option.

However, more often than not, the data collection was done just a few days before the competitions, which is why it is not ideal for all members to use the FSG Portal simultaneously. Simultaneous work could result in unsafe data handling, and the risk of deleting some crucial information is imminent, as there is no return button, and the deletion of entries can not be reversed.

The most common issue with the information collection was the inability to modify the "bought" or "made" status for the *parts*. If the invalid option was selected, the members were required to delete and re-enter the *part* once more.

As the Portal does not support the creation of the support file material, the team had to upload all related files to their server data storage and then create the support material file by hand, altering the header and footer for each inputted file.

That is why the team decided that creating a new system addressing all these issues and tailoring it to their unique requirements is the best possible course of action.

Chapter 4

Analysis

The goal of this chapter is to define the functional and non-functional requirements for the application and characterize them based on the MoSCoW method. These requirements are essential for developing use cases and use case diagrams. The chapter concludes with the domain class diagram of the application.

4.1 Requirements

Software requirements describe the services a software must provide and the constraints under which it must operate. Their collection helps us define the boundaries of the systems, enables a more accurate estimation of the workload, and captures the constraints placed on the system. [9]

Requirements are commonly divided into two main categories [10]:

- **Functional requirements** are statements of services that the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations.
- **Non-functional requirements** are constraints on the services or functions offered by the system, such as timing constraints, constraints on the development process, standards, and others.

The priority of the requirements is defined by the **MoSCoW method** [11]. It is a four-step approach to prioritizing which requirements have the best return on investment. The four categories are:

1. **Must Have** – the requirement is necessary for the successful completion of the project. Without it, the application will not function properly.
2. **Should Have** – the requirement should be included in the final project; however, it is not necessary for the first release.
3. **Could Have** – the requirement is not fundamental for the application; therefore, they are worked on only after completing all must-have and should-have requirements.
4. **Won't Have** – the requirement is optional and usually specifies future development of the application.

4.1.1 Functional Requirements

Functional requirements were defined based on the performed analysis. They take into account the requirements of the competition and the needs of the eForce Prague Formula team.

F1. **Systems overview (Must Have)**: The application will display a list of all systems defined by the rules.

F2. **System detail (Must Have)**: The application will allow user to click on a system and display additional information about said system.

F3. **Search bar (Could Have)**: The application will allow user to search for individual assemblies or parts based on their name or associated comment.

F4. **Assemblies overview (Must have)**: The application will allow user to view an overview of the assemblies belonging to a selected system.

F5. **Edit assembly (Must Have)**: The application will allow user to change the attributes of a particular assembly.

F6. **Delete assembly (Must Have)**: The application will allow user to delete an assembly.

F7. **Create new assembly (Must Have)**: The application will allow user to create a new assembly.

F8. **Create new subassembly (Must Have)**: The application will allow user to create a new subassembly.

F9. **View subassemblies (Must Have)**: The application will allow user to view a list of subassemblies falling under a selected assembly.

F10. **Edit subassembly (Must Have)**: The application will allow the user to change the attributes of a selected subassembly.

F11. **Delete subassembly (Must Have)**: The application will allow user to delete selected subassembly.

F12. **Parts overview (Must Have)**: The application will allow user to display an overview of the parts belonging to the respective assembly.

F13. **Create new part (Must Have)**: The application will allow user to create a new part, select its name, quantity, comment, and other necessary attributes.

F14. **File upload (Should Have)**: The application will allow user to upload a document for the selected part, assembly or sub-assembly.

F15. **File download (Should Have)**: The application will allow user to download a document for the selected part, assembly or sub-assembly.

F16. **Generate a document (Should Have)**: The application will allow user to download a document of the comrising the collected information.

F17. Drag and drop stamping (Could Have): The application will allow user to manually change the sorting in tables by dragging and dropping rows.

F18. Adding detail to part (Must Have): Ability to add processes, tooling, fasteners, materials and their costs to a part.

F19. Predefined macros (Could Have): The application will allow user to define processes, fasteners, materials and toolings macros including their name, comment and price.

F20. The cost of material relative to time (Won't Have): The application will allow user user to view the change of cost of material relative to time.

4.1.2 Non-functional Requirements

Non-functional requirements were defined based on the the performed analysis, and takes into account the infrastructure of the eForce Prague Formula team.

N1. Availability as a web application (Must Have): The application is available as a desktop web application.

N2. Authentication (Should Have): The application will enable to log in only to the authenticated user.

N3. ASP.NET core (Must Have): The application is developed using ASP.NET core based framework.

N4. MySQL Database(Must Have): The application needs to communicate with an instance of MySQL database.

N5. Containerization(Must Have): The application has to be built using a Docker container as a preparation for the application's deployment

4.2 Use Case Specification Models

Use-case specification models visually represent the actions between actors and the system and, therefore, represent the system's functional requirements in the context of the user's goals. The use cases and actors in the diagrams describe what the system does and how the actors use it, but not how the system operates internally. [12]

4.2.1 Actors

Before we can describe the use cases related to our system, we first need to predefined the **actors**. An actor represents a role of a user that interacts with the system. The user can be a human user, an organisation, a machine, or another external system. [13]

The implemented application will have one user, the user being a member of a Formula Student team who will use the application for collection of cost and manufacturing data.

4.2.2 List of Use Cases

UC1. **Log in** – The web application presents the user with log in form. The user fills in the fields for username and password and then clicks on the button to proceed with login. If the username and password are correct, the user is logged in into the application. Otherwise the user is advised to inform the admin from the team to resolve the issue (a.k.a. proceed with registration or update their forgotten password).

UC2. **View systems** – The application presents user with a list of systems in a form of a table. Each system represents one row of the table.

UC3. **Manage assemblies** – Specified in the Figure 4.2.

UC4. **Manage subassemblies** – Specified in the Figure 4.3.

UC5. **Manage parts** – Specified in the Figure 4.4.

UC6. **Generate document** – The user clicks the ”generate cost report” button in the top left corner of the screen after which the application generates a document containing all the systems, its assemblies, subassemblies, parts and all its related meta data and creates a document which then the user can download using the ”download” button with which he is presented.

UC7. **Log out** – The option to log out is presented only to a logged in user. The button for log out is located in the bottom left corner of the screen. After the user clicks the button, he is logged out and redirected to the log-in screen.

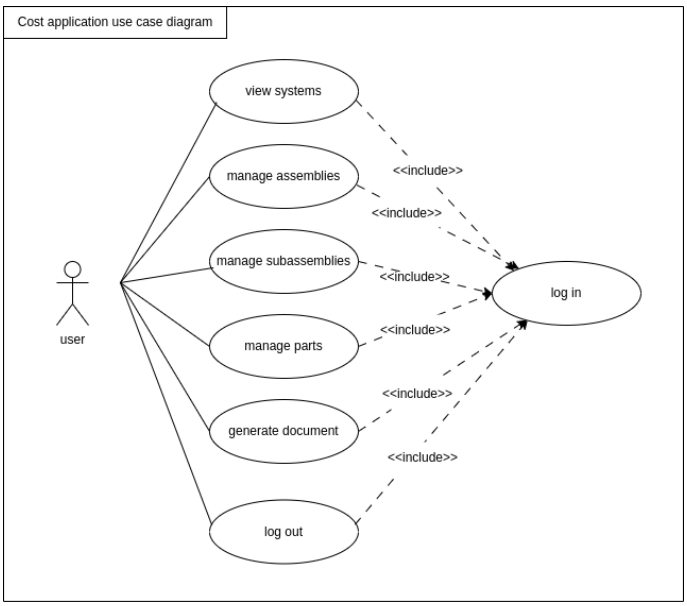


Figure 4.1: General use cases, created using [14]

UC8. **Find assembly** – The application presents the user with a table containing all the assemblies from certain system. The user can then locate the assembly in the table manually, or he can use the search bar in the top of the screen to look up the assembly either by name or by its comment.

UC9. **View assemblies** – The user chooses a system, whose assemblies he would like to display, then locates the system in the systems table and clicks on the button ”assemblies”. The application redirects the user to a new page with all the assemblies from the chosen system. The user can also search among the assemblies based on its ”name” or ”comment” by clicking on the search bar at the top of the screen.

UC10. **Create new assembly** – The user clicks on the button ”create new assembly” located in the top of the screen. He is then presented with a pop up form in which he has to fill out the ”name” of the assembly. If he decides to not proceed with creation of new assembly he can click the ”cancel button”. The system then asks the user if he doesnt want to proceed and if so the pop up form is then closed. Otherwise the user choses the button ”create” and the pop up form closes and the new assembly is created.

UC11. **Delete assembly** – The user locates the assembly he would like to delete in the assemblies table and clicks on the ”delete” button in the table row. The user is then presented with a confirmation dialog which asks him if he truly wants to delete the assembly. The user then can proceed with the deletion by clicking the ”yes” button, which triggers the deactivation of the assembly and all the subassemblies and parts which are included in the subassembly and the confirmation dialog closes. If the user does not wish to proceed with the deletion he can click the ”no” button and the confirmation dialog closes and the assembly remains unchanged.

UC12. **Edit assembly** – The users locates the assembly he wishes to edit in the assemblies table and clicks on the ”edit” button in the corresponding row. The application then shows the user an edit form in which the user can change the assembly’s ”name” and/or ”comment”. Afterwards he can click on the ”submit” button and the changes will be applied and the edit form closes. Otherwise the user can click on the ”cancel” button, the edit form closes and the assembly remains unchanged.

UC13. **Upload document** – The user locates the assembly to which he wishes to upload the document and finds it in the assemblies table. Then on the corresponding row he clicks on the ”upload a document” button. Application then shows him the upload form in which he will upload the document. He can then proceed to click the ”save” button which will save the file and the upload form closes. Otherwise the user can click the ”cancel” button which will trigger the confirmation dialog in which it asks the user wheater he wishes to cancel the upload. If the user than clicks on the ”yes” button, both the confirmation form and the confirmation dialog closes and no changes are made to the

application. Otherwise the confirmation dialog closes and the user can proceed with the upload.

UC14. Download document – The user locates the assembly whose files he wishes to download and finds it in the assemblies table. Then on the corresponding row he clicks on the ”download a document” button. The application then proceeds to download the attached documents.

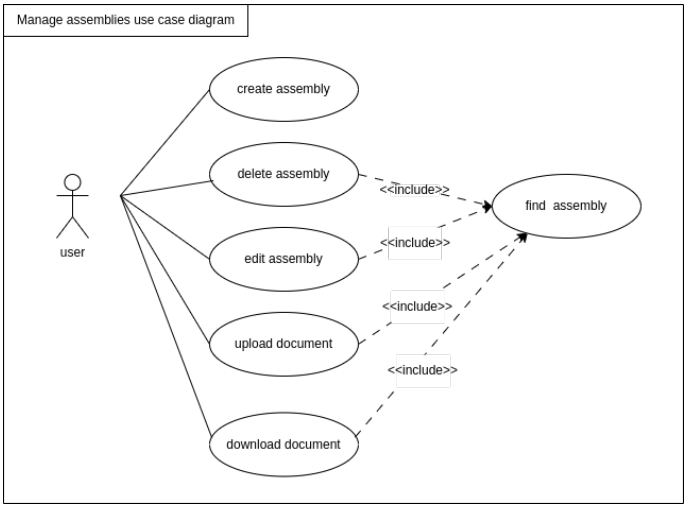


Figure 4.2: Manage assemblies use case diagram, created using [14]

UC15. Find subassembly – The application presents the user with a table containing all the subassemblies from certain system/assembly. The user can then locate the subassembly in the table manually, or he can use the search bar in the top of the screen to look up the subassembly either by name or by its comment.

UC16. Create subassembly – The user clicks on the button ”create subassemly” located in the top of the screen. He is then presented with a pop up form in which he has to fill out the ”name” of the subassembly. If he decides to not proceed with creation of new subassembly he can click the ”cancel” button. The system then asks the user if he doesn't want to proceed and if so the pop up form is then closed. Otherwise the user chooses the button ”create” and the pop up form closes and the new sub-assembly is created.

UC17. Edit subassembly – The users locates the subassembly he wishes to edit in the subassemblies table and clicks on the ”edit” button in the corresponding row. The application then shows the user an edit form in which the user can change the subassembly's ”name” and/or ”comment”. Afterwards he can click on the ”submit” button and the changes will be applied and the edit form closes. Otherwise the user can click on the ”cancel” button, the edit form closes and the subassembly remains unchanged.

UC18. View subassemblies – The user chooses an assembly , whose sub-assemblies he would like to display, then locates the assembly in the assemblies table and clicks on the button ”sub-assemblies”. The application redirects the user to a new page with all the parts from the chosen assembly . The user can also search among the sub-assemblies based on its ”name” or ”comment” by clicking on the search bar at the top of the screen.

UC19. Delete subassembly – The user locates the subassembly he would like to delete in the subassemblies table and clicks on the ”delete” button in the table row. The user is then presented with a confirmation dialog which asks him if he truly wants to delete the subassembly. The user then can proceed with the deletion by clicking the ”yes” button, which triggers the deactivation of the subassembly and all the parts included in it and the confirmation dialog closes. If the user does not wish to proceed with the deletion he can click the ”no” button and the confirmation dialog closes and the subassembly remains unchanged.

UC20. Upload document – The user locates the subassembly to which he wishes to upload the document and finds it in the subassemblies table. Then on the corresponding row he clicks on the ”upload a document” button. Application then shows him the upload form in which he will upload the document. He can then proceed to click the ”save” button which will save the file and the upload form closes. Otherwise the user can click the ”cancel” button which will trigger the confirmation dialog in which it asks the user weather he wishes to cancel the upload. If the user than clicks on the ”yes” button, both the confirmation form and the confirmation dialog closes and no changes are made to the application. Otherwise the confirmation dialog closes and the user can proceed with the upload.

UC21. Download document – The user locates the subassembly whose files he wishes to download and finds it in the subassemblies table. Then on the corresponding row he clicks on the ”download a document” button. The application then proceeds to download the attached documents.

UC22. Find part – The application presents the user with a table containing all the parts from certain system /assembly/subassembly. The user can then locate the part in the parts table manually, or he can use the search bar in the top of the screen to look up the parts either by name or by its comment.

UC23. Create part – The user clicks on the button ”create part” located in the top of the screen. He is then presented with a pop up form in which he has to fill out the ”name” of the part, its “comment”, “quantity” and checks wheater it is custom made or bought. If he decides to not proceed with creation of new part he can click the ”cancel” button. The system then asks the user if he doesn’t want to proceed and if so the pop up form is then closed. Otherwise the user chooses the button ”create” and the pop up form closes and the new part is created.

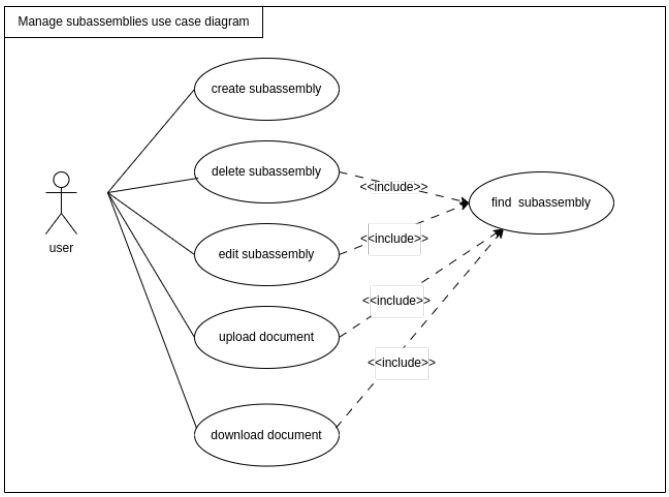


Figure 4.3: Manage subassemblies use case diagram, created using [14]

UC24. Edit part – The users locates the part he wishes to edit in the parts table and clicks on the ”edit” button in the corresponding row. The application then shows the user an edit form in which the user can change the assembly's ”name” and/or “quantity”, “custum made/ bought”. Afterwards he can click on the ”save changes” button and the changes will be applied and the edit form closes. Otherwise the user can click on the ”cancel” button, the edit form closes and the assembly remains unchanged.

UC25. Delete part – The user locates the part he would like to delete in the parts table and clicks on the ”delete” button in the table row. The user is then presented with a confirmation dialog whis asks him if he truly wants to delete the part. The user then can proceed with the deletion by clicking the ”yes” button, which triggers the deactivation of the part and the confirmation dialog closes. If the user does not wish to proceed with the deletion he can click the ”no” button and the corfirmation dialog closes and the part remains unchanged.

UC26. View parts – The user chooses a system or an assembly or an subassembly, whose parts he would like to display, then locates the system or assembly or subassembly in the systems or assemblies or subassemblies table and clicks on the button ”parts”. The application redirects the user to a new page with all the parts from the chosen system or assembly or subassembly. The user can also search among the parts based on its ”name” or ”comment” by clicking on the search bar at the top of the screen.

UC27. Upload document – The user locates the part to which he wishes to upload the document and finds it in the parts table. Then on the corresponding row he clicks on the ”upload a document” button. Application then shows him the upload form in which he will upload the document. He can then proceed to click the ”save” button which

will save the file and the upload form closes. Otherwise the user can click the ”cancel” button which will trigger the confirmation dialog in which it asks the user wheater he wishes to cancel the upload. If the user than clicks on the ”yes” button, both the confirmation form and the confirmation dialog closes and no changes are made to the application. Otherwise the confirmation dialog closes and the user can proceed with the upload.

UC28. Download document – The user locates the part whose files he wishes to download and finds it in the parts table. Then on the corresponding row he clicks on the ”download a document” button. The application then proceeds to download the attached documents.

UC29. Add details – The user locates the part to which he wishes to add additional details. Then clicks on the button “DBOM”. This will open up a form in which the user chooses : the “type” of detail (it can be material / fastener/ process / tooling), its “quantity”, “costs (per piece)”, “comments”, “cost comments”. If he decides to not proceed with creation of new detail he can click the ”cancel button”. The system then asks the user if he doesn't want to proceed and if so the pop up form is then closed. Otherwise the user chooses the button ”create” and the pop up form closes and the new detail is created.

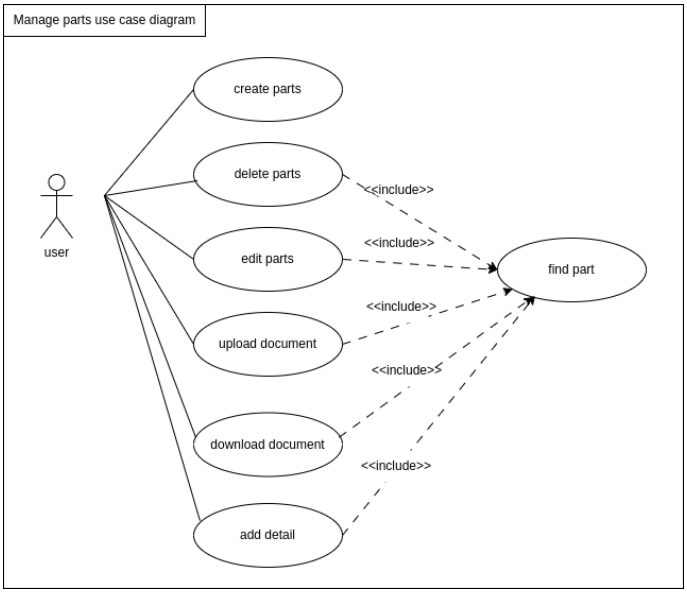


Figure 4.4: Manage parts use case diagram, created using [14]

4.3 Domain conceptual model

The domain conceptual model provides a visual representation of the static structure of the system. It describes the elements of the system and the relationships between them. These elements are represented by classes and can

characterise, for example, persons, things, events, or abstract concepts such as groups.[15]

Entities and their relationships as well as the final diagram shown in Figure 4.5 was constructed based on the rules of the completion as well as the analysis conducted in this chapter.

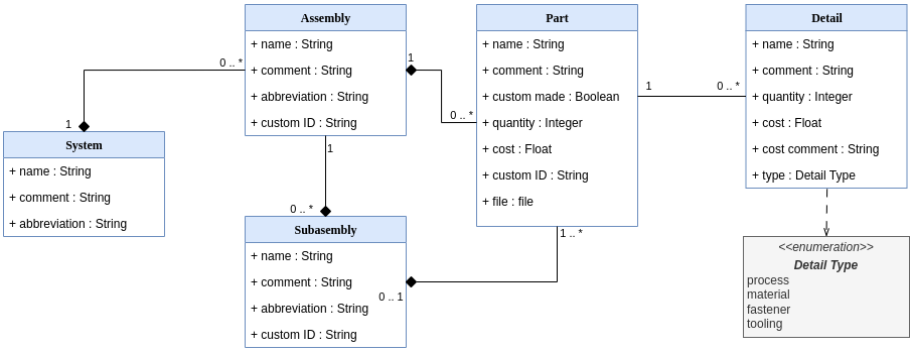


Figure 4.5: Domain conceptual model, created using [14]

4.3.1 System

Entity system represents a system defined by the rules, that has its name, two letter abbreviation used for generating of custom IDs and non-required comment describing the system further.

4.3.2 Assembly

Entity assembly represents an assembly as described by the rules. It has its name, non-required comment, three letter abbreviation used for generating of custom ID. The name can be chosen from a predefined Table A.2 or created, however all created names have to start with a prefix ”Other:”.

4.3.3 SubAssembly

Entity subassembly represents a smaller subgroup of parts included into an assembly. It has its name, three letter abbreviation and a non-required comment.

4.3.4 Part

Entity Part represents any part which is used for the manufacturing of a monopost. It has its name, comment, indication whether it was custom made or bought, quantity, and also a generated custom ID based on the system, assembly and subassembly it belongs in.

4.3.5 Detail

Entity detail further specifies the part. It has its name, comment, quantity, cost, which specifies the Price per piece , cost related comments and detail type which has to be one of the predefined types.

Chapter 5

Design

In this chapter i will present all the technologies used in implementation of my application as well as brief overview of the used dabase and user interface libraries.

5.1 .NET Development

During the mid-1990s, in response to ever so growing popularity of Java, Microsoft decided to develop the **.NET Framework**. The first version, released in 2002, enabled to build and execute applications in almost 22 different programming languages. However, it did have a significant drawback in that it was limited to only Windows environment. [16]

In pursuit of a genuinely cross-platform version of the .NET Framework, Miguel de Icaza and Nat Friedman established the **Mono Project** in 2004, which aimed to develop a Linux-compatible version of .NET Framework. Despite the emergence of other third-party implementations, Microsoft remained focused on its Windows-based version [16]

That changed in 2016 when Microsoft presented **.NET Core**, a cross-platform runtime available on Windows, macOS, and Linux, which became faster and more modular than its predecessor. The Last significant change came in 2020, when a new implementation of .NET was released, dropping the Core naming scheme. Modern .NET supports all types of development, the three most common being web, desktop, and mobile applications.

Alongside the .NET Core release, Microsoft also introduced **ASP.NET core** framework for the development of interactive web applications. ASP.NET Core provides a variety of programming models, such as Web API, MVC, Radzen, or Blazor. [17]

The decision to select .NET for the application development was motivated by the collective expertise in C# and .NET within the team's IT department. As a result, it enhances the team's ability to address issues and implement future enhancements swiftly, improving the application's scalability and maintainability.

5.1.1 Blazor

Blazor is an open-source web-development framework based on ASP.NET core that enables to develop web applications using C#, CSS and HTML instead of JavaScript, the industry's more common alternative. As of .NET6 which is used in the developed application, Blazor provides two hosting models for purely web applications. [18]

5.1.2 Blazor Server

Blazor server application runs entirely on the server using SignalR, an open-source library providing real-time web functionalities [19], to connect to the user interface, as shown in Figure 5.1a. All changes on the client side are then communicated back to the application using SignalR, where they are processed by the server.

The **advantages** of Blazor server are: [20]

+ High speed thanks to the pre-rendering of HTML component.
+ Accessibility through various web browsers, even older versions.
+ Security as a result of the whole business logic running on the server, therefore not exposing its inner workings to the client.

The **disadvantages** of Blazor Server are: [20]

– No offline support caused by the need for a live SignalR connection.
– High latency as a result of too many calls made to the server.

5.1.3 Blazor WebAssembly

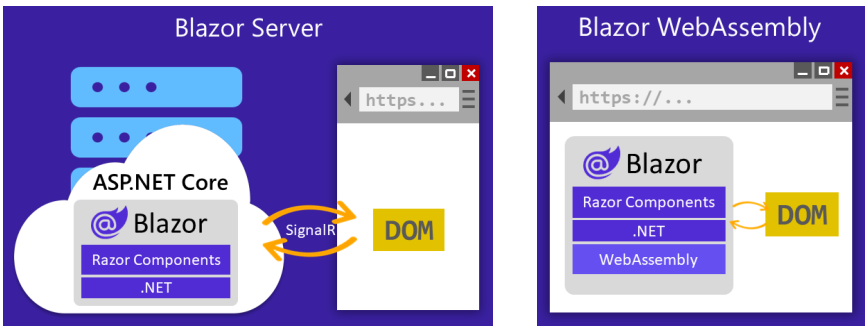
Blazor Web Assembly also refered to as Blazor WASM downloads all necessary components and dependencies to the browser and runs them from there, as shown in Figure 5.1b. That eliminates the need for live connection to the server and in return provides faster and more responsive user interface with offline support. [21]

The **advantages** of Blazor Web Assembly are: [20]

+ Offline support.
+ Faster user interface as a result of minimal interactions with the server.

The **disadvantages** of Blazor Web Assembly are: [21]

– The performance of the application is limited only to the capatibilities of the browser.
– It cannot connect to database directly due to security restrictions and sandboxing limitations of web browser and therefore requires an API or other form of serverver connection.
– The initial download to the browser can be slow if the application is larger in size.



(a) Blazor Server (b) Blazor WebAssembly

Figure 5.1: Comparison of Blazor Server and Blazor Web Assembly hosting models. [22]

5.1.4 Chosen Hosting Model

After reviewing the advantages of both hosting models, it was decided to implement a Blazor Server application. Offline support for the application is not a priority at this moment; hence, it was not a deciding factor. It is also desirable to interact with the database and file storage directly. Blazor Server meets these requirements and provides us with the opportunity to develop a genuinely full-stack solution without the need for any API implementation.

5.2 Database

As the primary goal of this application is to collect and categorize data, it is natural and almost vital to use appropriate data storage, in this case, in a Relational Database Management System (RDBMS).

Entity Framework is an object-to-data mapping techology for .NET applications. It enables programmers to read, alter, and query the database without using almost any SQL, only using the entities of the program and LINQ queries. [23]

There are three approaches when using EF Core: [24]

- **Database First**: When using this approach, the database already exists, and EF is used to create a .NET Model depicting the database schema.
- **Model First**: The database does not yet exist. Therefore, the model is built first, and then the EF generates entities and a database script.
- **Code First**: The dabase also does not exist. The application entities are built first, and from them a database is created using migrations.

The wForce Prague Formula team currently uses a MySQL database to store its data; therefore, the implementation will stick to this provider. The production database will, however, not be used for the development because it currently does not include any schema related to the Cost & Manufacturing event, and therefore, a code-first approach using Entity Framework will be implemented. It was also opted against using the production database directly to avoid corrupting other data stored in the database.

5.3 Architectural Principles

Each application requires a well tought out architecture to make the application cleaner and more mantainable in the future. [25]

A guiding principle in architecture development is the separation of concerns. Each part of the application should have its predefined purpose, and no two parts should share the same processes, therefore following the Do not Repeat Yourself architectural principle. Duplication of code not only worsens the maintainability of the application but also adds to its technical debt. [26]

Encapsulation is essential for all parts of the application to define the interfaces through which they communicate with other components. This practice enables adjustments to the internal workings of a component without affecting the syntactical correctness of dependent components [27].

Based on the Explicit dependencies principle, the classes should explicitly require any object they need for proper functionality rather than relying on a declaration of global dependancies. [25]

The application should adhere to the Dependency Inversion principle in which rather than classes being dependent on other concrete functions and objects, a dependence on abstract interfaces is implemented. If the classes are dependent on concrete implementation, it results in a highly coupled system, with each module directly reliant on the lower modules and therefore limiting the future scalability of the application. [28]

5.3.1 Proposed architecture

Based on the defined architecture principles, the application will implement a layered architecure pattern. Layered architecture is an software application design which divides the application resources into horizontal layers. Each layer is responsible for certain type of operation and only communicates with layers directly above or directly below itself. [29] This approach achieves high modularity and test-ability of the application. To follow the Dependncy inversion principle both buisness layer and data acess layer will implement abstract interfaces and their concrete implementations.

The application will contain following layers as shown in the diagram 5.2:

1. **Presentation** layer, represents the user interface of the application.
2. **Business** layer, represents services which contain buisness logic of the application and handles exceptions.
3. **Data Access** layer, represents data models and repository, that will interact with the database layer.
4. **Database** layer, represents the database.

5.4 Authentication

Authentication is the process of confirming the identity of a user, device, or system. Authentication is commonly used in various contexts, such as accessing computer systems, online accounts, networks, or physical locations. [30] We can distinguish two types of authentification: stateful and stateless.

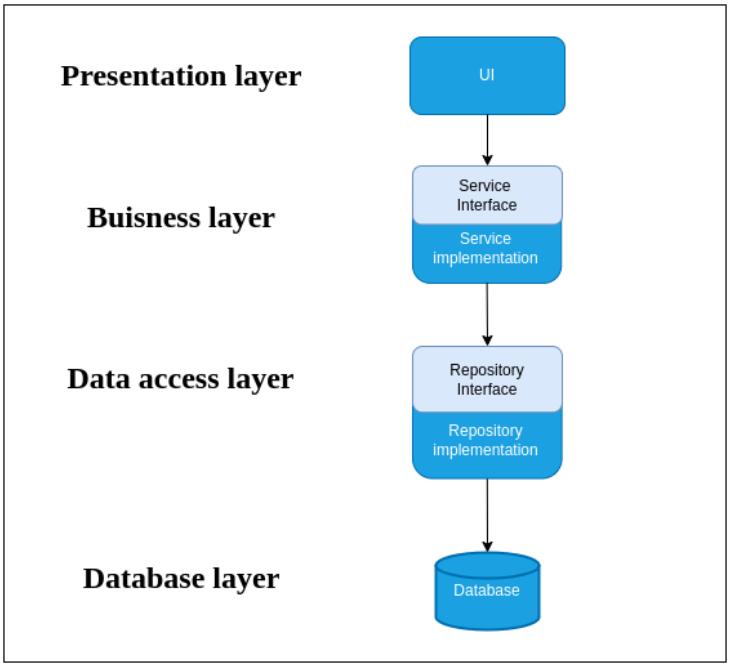


Figure 5.2: Layered architecture of the application. Created using [14]

5.4.1 Stateful Authentication

Stateful authentication involves the server retaining the session state for each authenticated user by generating a unique session identifier. This allows subsequent requests from the same user to be associated with their session data stored on the server. While it provides features such as session management, server-side security, and user tracking, it may lead to increased server overhead, scalability challenges, and added complexity in session management. [31]

5.4.2 Stateless Authentication

Stateless authentication uses tokens to contain all necessary authentication information, eliminating the need for the server to maintain a session state. It simplifies server architecture and scales more quickly but requires careful token management and lacks specific session management capabilities, potentially introducing security vulnerabilities if not implemented correctly. [31]

5.4.3 LDAP Server

The team eForce Prague Formula operates a custom LDAP server for their authentication needs. LDAP, or Lightweight Directory Access Protocol, is a data access and maintenance protocol primarily used for user authentication and authorization.

For the purpose of the developed application, LDAP authentication will be combined with cookie-based session management for a seamless user experience. This approach ensures secure user access while taking advantage of the efficiency of cookie-based sessions. The integration also provides scalability and

flexibility in managing user sessions, enabling tailored features such as session expiration and access controls to suit the application's requirements.

5.5 File Handling

To ensure that the implemented application can support upload, download, and overall management of files, it is necessary to implement a form of file management.

5.5.1 File Storage approaches

The three most common file management approaches are described in the following subsections based on [32].

5.5.1.1 Database

In this strategy, files are transformed into a binary stream and are subsequently stored directly within the database. This method proves favourable for managing small-scale file uploads, as it maintains efficient file handling speed and potentially saves costs by not needing to invest in third-party cloud solutions. However, it is unsuitable for any larger files as it can increase the database's size significantly.

5.5.1.2 Physical Storage

This approach stores data in a predefined storage such as a hard drive. It is ideal for larger file quantities as it poses fewer size limitations as opposed to the database approach. It is, however, necessary to ensure that correct directory permissions are in place. This option is also strained by the need for physical storage, which is only available in some situations.

5.5.1.3 Cloud

Cloud services propose all the advantages of physical storage solutions but have one significant advantage: they require no physical hardware. That is why it is ideal for companies with fewer on-premisses resources. The only disadvantage is that the majority of these services are not free of charge and, therefore, can impact the project's financial budget.

5.5.1.4 Chosen Variant

The team operates two servers, one running on Debian GNU/Linux 7 and the other on Fedora Linux; therefore, it is equipped with satisfactory physical storage, and the cloud variant is unnecessary. Physical storage will, therefore, be implemented, resulting in zero-cost implementation.

5.5.2 Security

Few security measures will be implemented to ensure higher security while handling files. Nominally:

- Only files of type .pdf, .jpg, .png and .svg can be uploaded.
- The number of files and file size will be restricted to prevent insecure file upload attacks.
- The uploaded files will be renamed to prevent overwrite and path traversal attacks.

5.6 User Interface

The last phase of application design involves drafting the user interface, a crucial element that significantly impacts user workflow. UI design contains strategic decisions regarding layout, fonts, colors, and overall styling to ensure intuitive usability. The most effective UI designs are consistent and simple, providing a seamless user experience. [33]

5.6.1 UI Libraries

There are many possible user interface libraries on today's market which help enhance the appearance of Blazor Applications. To determine the best possible solution for the to be implemented application, the analysis of all possible solutions was made. The main requirement for any chosen solution was to have open-source access, as the open-source solutions are more then satisfactory. Other requirements where, expandable data grid with supported grouping of the data, form fields component with drop down, auto complete and their validation, file inputs and alert dialog windows.

As a result 3 open-source libraries where chosen – Radzen Blazor [34], Blazorize [35] and Mud Blazor [36], all of which fulfil the above mentioned criteria. In the end it was decided to implement Radzen Blazor library, purely based on its design and appearance.

5.6.2 UI Design

The design includes a distinctive red colour, one of the two official colours of the eForce Prague Formula team. *Central Information System* is the official name of the application, chosen by the team members, and is officially referred to by its acronym *CIS*.

5.6.2.1 Authentication

Upon launching the application, the user is presented with a login form for authentication. The login form features the previously mentioned red colour scheme and includes the team's logo and the application name on the left-hand side. After the user's identity has been verified, he is then redirected into the application. Notably, the screen does not include a registration button, as the system is internal to the team, and registration is only facilitated directly by the IT department.

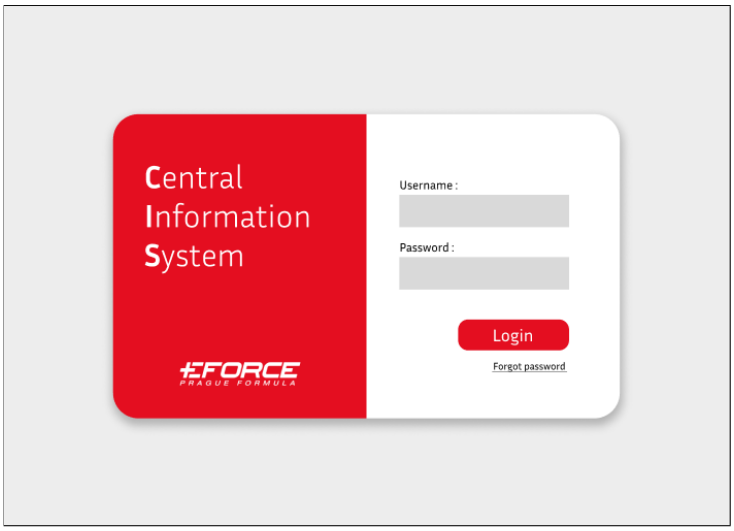


Figure 5.3: Login form design, created using [37]

5.6.2.2 Navigation Bar

All screens within the application, except for the login screen, incorporate a navigation sidebar positioned on the left-hand side. This sidebar comprises the application name displayed at the top, followed by buttons that enable navigation throughout the application. Additionally, at the bottom of the sidebar, users can find a logout button along with the team's logo.

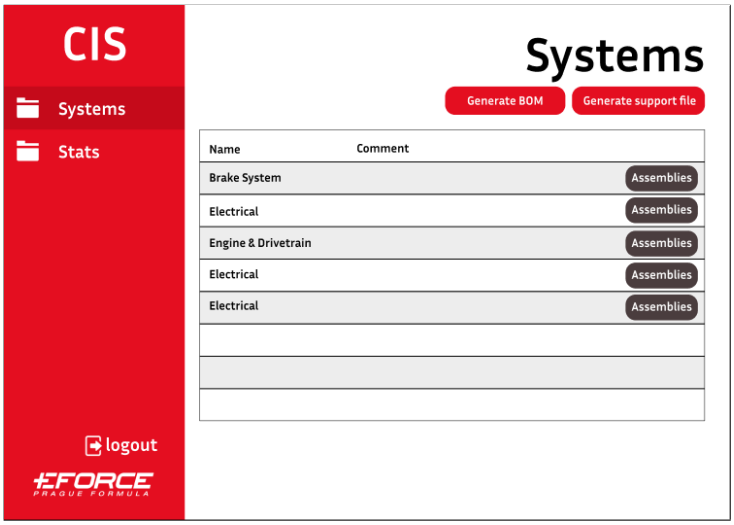


Figure 5.4: Systems page design, created using [37].

5.6.2.3 Systems Overview

The initial home screen showcases a table listing various systems per competition specifications. Each entry in the table includes the system's name,

abbreviation, and a button to reveal further details about its assemblies and parts. Positioned above the table are two buttons: the "Generate BOM" button, which generates and downloads a file containing the entire data structure from systems to parts' details, containing all database entries, and the "Generate Support File" button, which combines all uploaded files and downloads it automatically.

5.6.2.4 Assemblies Overview

Upon selecting a system, the user is redirected to a page that displays the corresponding assemblies of the system. These assemblies include also subassemblies, if they have any. Each entry in the table includes the assemblies's name, comment, custom ID, and four buttons for downloading of files, editing, deletion and routing to the corresponding parts. In the top right corner of the screen is located an "add assembly" button which enables the user to create new assembly.

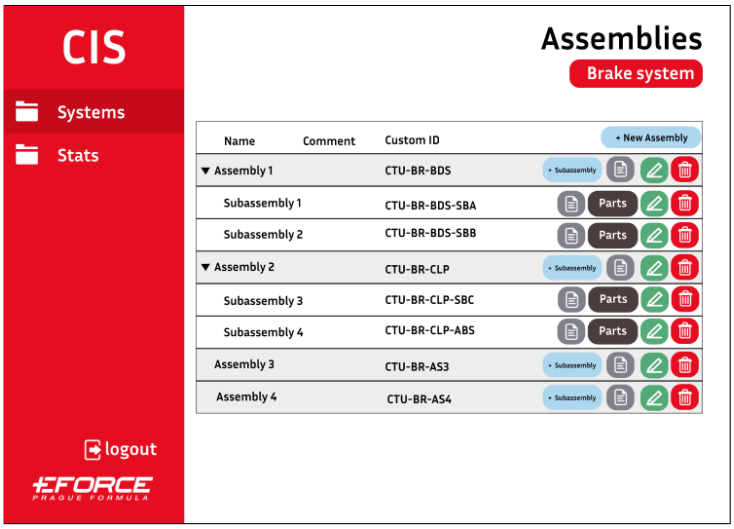


Figure 5.5: Assemblies table design, created using [37].

5.6.3 Parts Overview

Upon selecting an assembly (or subassembly), the user is redirected to a page that displays the corresponding parts of the assembly (or subassembly). Each entry in the table includes the part's name, comment, custom ID, quantity, and four buttons for downloading of files, managing the part's details, editing, and deletion. In the top right corner of the table, there is a button labeled "Add Part" to create used to create new part.

5.6.3.1 Parts Detail Overview

When the user clicks on the "Part Details" button, a dialog box will appear, presenting a table of all the details of the parts. Each entry in the table includes the part's name, comment, cost per piece, cost-related comment, and quantity.

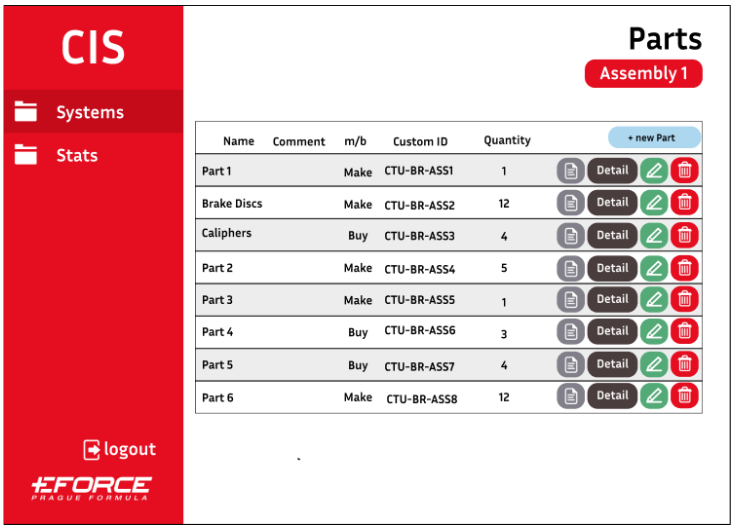


Figure 5.6: Parts table design, created using [37].

Additionally, two buttons are provided for editing and deleting the entry. A button to create a new part is at the top right corner of the table. Users can close the dialog by clicking the ”X” in the top right corner or the close button in the bottom right corner.

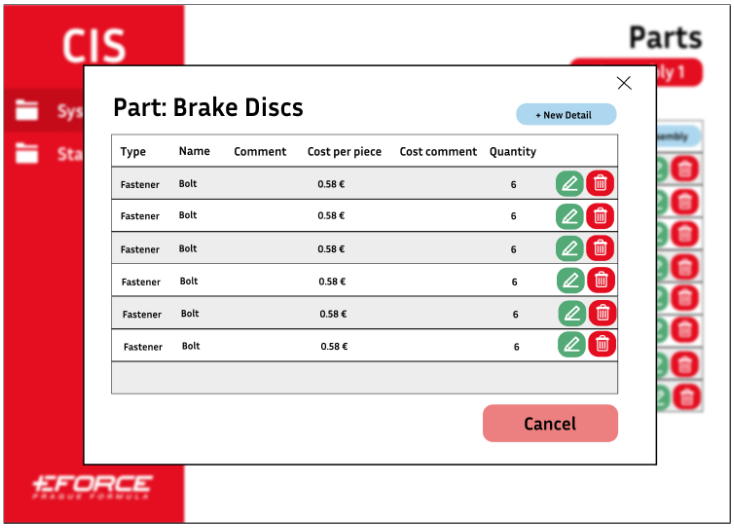


Figure 5.7: Parts detail page, created using [37].

5.6.4 Add & Edit form

Both the add and edit forms follow the same design pattern, differing only in the input or edited fields. Upon selecting the ”Add” or ”Edit” button, a dialog window opens, displaying all relevant fields. At the bottom right corner of the dialog, two buttons are provided: one for submitting the entry and the other

to close the dialog. Users can also close the dialog by clicking the ”X” in the top right corner.

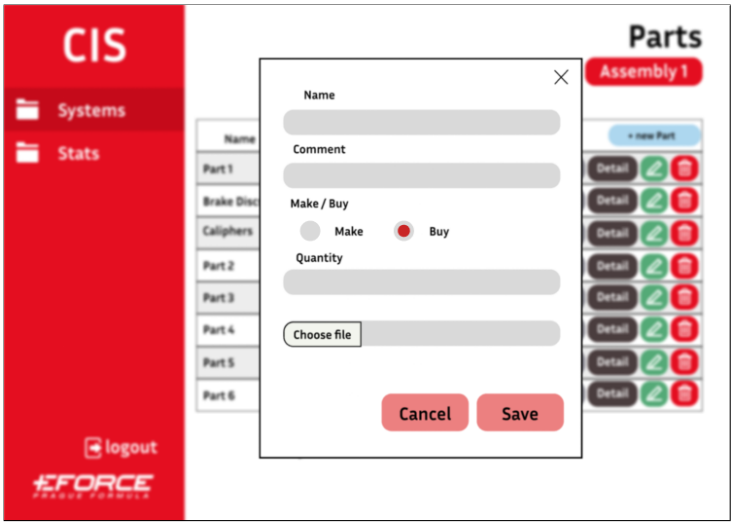


Figure 5.8: Add/edit form, created using [37].

Chapter 6

Implementation

This chapter provides a closer look into the factual implementation of the application, based on the findings from the previous chapters on analysis and design . The development and operation of the process will be presented in detail, along with noteworthy design choices and functionalities.

6.1 Software Project Managment

Every software project requires effective management to ensure high-quality work output and efficient use of time. The size of the development team determines the scale of management needed. [38] Although a single person developed the application, it is essential to implement tools such as version control, as the application will be maintained by a development team from eForce Prague Formula in the future. Having these tools in place ensures a smooth collaboration, efficient code management, and an easy onboarding process for new team members.

6.1.1 Data Versioning

Data versioning is one of the most critical aspects of project management. It enables the development team to monitor and manage code changes. [39] One of the most commonly used tools for data versioning is GitLab.

GitLab is an open-source platform used to integrate developed software. It functions as a centralized hub for incorporating various DevOps tools into the codebase, including CI, CD, and issue management, consolidating these functionalities within a single platform. [40]

6.1.2 Documentation

The application was documented using XML comments throughout its data-handling functions. This embedded documentation provides insights into each function's purpose and behavior, aiding in troubleshooting and future development. A README file was also included, containing snippets of implementation decisions described in this thesis. This additional information offers further context without the need to review the entire thesis.

6.1.3 Containerization

Containerization is a process of virtualizing operating systems. It involves encapsulating an application and all its dependencies into a container that is isolated from other applications and creates a version of a virtual environment. [41]

Docker is a platform that helps developers quickly create, deploy, and run containers. [42] Team eForce also uses Docker containerization for effective application deployment into production. The container is set up based on its Dockerfile, which specifies the configuration of the container. An illustration of the Dockerfile used can be viewed in Listing 1.

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0-alpine AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0-alpine AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["CIS_app/CIS_app.csproj", "CIS_app/"]
RUN dotnet restore "CIS_app/CIS_app.csproj"
COPY . .
WORKDIR "/src/CIS_app"
RUN dotnet build "CIS_app.csproj" -c $BUILD_CONFIGURATION -o
↪  /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "CIS_app.csproj" -c $BUILD_CONFIGURATION -o
↪  /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "CIS_app.dll"]
```

Listing 1: Multistage Dockerfile used in the implementation

6.1.4 CI/CD

Continuous Integration (CI) and Continuous Delivery (CD) are important steps of DevOps methodology that help simplify and automatize the software development process by seamlessly integrating code changes and ensuring their delivery to production environments. [43]

CI a development technique centered on the periodic and automated integration of code changes into a remote repository or main branch. The primary objective is to minimise disparities between changes made in developers' local environments. CI can also facilitate building and testing the application for each integration. It offers prompt feedback in case of build failures or test errors, promptly notifying the development team. [43]

CD is a subsequent step following the CI. The role of CD is to deploy the application into production without any manual interference. [43]

Gitlab CI is executed through so-called *pipelines*, which represent a series of tasks known as *jobs*, which can be additionally grouped into *stages*. These *stages* are then executed sequentially. The pipeline can be executed with each change made in the codebase or triggered in periodic intervals. The pipeline is written in YALM format and is located in the `.gitlab-ci.yml` file within the gitlab repository. [44]

The pipeline used in the implementation can be seen in Listings 2 to 5.

```
image: docker:20.10.16

variables:
  DOCKER_HOST: tcp://docker:2375
  DOCKER_TLS_CERTDIR: ""

stages:
  - build
  - test
  - publish

...
```

Listing 2: CI/CD pipeline initialization and stages declaration

Listing 2 represents the begining of CI pipeline. This section specifies the Docker image used, in this case, version 20.10.16. It outlines the required variables, including the remote connection to the Docker daemon accessible via TCP port 2375. Another variable indicates that transport layer security is disabled. The pipeline consists of three stages that will be executed one after the other. The following paragraphs provide more details on each stage.

In Listing 3, the job named `docker-build` is showcased as part of the build phase. It begins by specifying the Docker `20.10.16-dind` service, which represents a version of Docker in Docker. This service enables the execution of Docker commands in a CI/CD environment. Subsequently, the job defines several variables, including the Docker registry where images will be pushed, `IMAGE_TAG` (a tag for an image using the registry URL and a unique identifier based on the commit SHA), and `LATEST_TAG` (a tag for the latest version of the Docker image). Before executing the main script, the job initiates authentication for the provided Docker registry using `CI_REGISTRY_USER` and `CI_REGISTRY_PASSWORD`, which are GitLab variables storing factual credentials to prevent direct exposure in the pipeline. Upon successful authentication, the latest Docker image is pulled from the registry, followed by the build of a Docker image using the Dockerfile located in the `Cis_app` directory. The image is then tagged with both `IMAGE_TAG` and `LATEST_TAG`. The final command in this stage involves pushing the newly built image into the Docker registry.

In Listing 4, the test stage of the CI/CD pipeline is showcased. The stage first specifies the Docker image used for this stage, enabling the execution of the C# application written in the .NET environment. The job then redirects

```
docker-build:
  stage: build
  services:
    - docker:20.10.16-dind
  variables:
    REGISTRY_URL: eforce1.feld.cvut.cz:4567
    IMAGE_TAG: $REGISTRY_URL/ritzkvik/cis_app:$CI_COMMIT_SHORT_SHA
    LATEST_TAG: $REGISTRY_URL/ritzkvik/cis_app:latest
  before_script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login -u
    ↪  "$CI_REGISTRY_USER" --password-stdin $REGISTRY_URL

  script:
    - docker pull $LATEST_TAG || true
    - docker build --cache-from $LATEST_TAG -t $IMAGE_TAG -t
    ↪  $LATEST_TAG -f CIS_app/Dockerfile  .
    - docker push $IMAGE_TAG
```

Listing 3: Build stage in CI/CD pipeline

```
test:
  stage: test
  image: mcr.microsoft.com/dotnet/sdk:6.0
  before_script:
    - cd CIS_app_testing
  script:
    - dotnet test
```

Listing 4: Test stage in CI/CD pipeline

into the `CIS_app_testing` project, which includes unit tests, further described in Chapter 7.1. The tests are executed using the `dotnet test` command.

In the Listing 5, the publish stage of the CI/CD pipeline is showcased. The prerequisite for the stage execution is the successful completion of the docker-build job. The docker-published job is marked as manual and, therefore, requires the job to be triggered manually, which is the favored variant, as this step is necessary only prior to application deployment. The script first pulls the Docker image specified by the `IMAGE_TAG` variable. It tags the image with the `LATEST_TAG` and pushes the image back into the docker registry.

With the completion of the docker-publish stage, the application is now fully prepared for the future deployment to the server.

6.2 Design Patterns

Design patterns are standard practice in software design used to combat reoccurring problems and improve the application's modularity and cohesion.

In the implemented application, there are three recognizable design patterns.

```
docker-publish:
  stage: publish
  extends: docker-build
  needs:
    - docker-build
  rules:
    - when: manual
  dependencies:
    - docker-build

  script:
    - docker pull $IMAGE_TAG
    - docker tag $IMAGE_TAG $LATEST_TAG
    - docker push $LATEST_TAG
```

Listing 5: Publish stage in the CI/CD pipeline

6.2.1 Repository Pattern

As indicated by the analysis, the repositories pattern aligns with the principles of layered architecture.The repositories facilitate communication with the database to extract information, effectively comprising a data access layer. Notably, repositories refrain from enforcing business logic or handling exceptions.

The layer is divided into interfaces and implementations to improve the abstraction and testability of the application. This segmentation enables efficient mocking of interfaces, as demonstrated in Chapter 7.

In C#, the repository pattern is implemented by defining `interfaces`, which are then implemented by concrete `classes`. The pattern implementation is shown in Listing 6. All repositories exclusively contain CRUD operations (Create, Read, Update, Delete) for database queries.

6.2.2 Service Pattern

Similarly to the application's repositories, the application also implements a service design pattern. Services are part of the application's business layer and contain all necessary business logic, including handling exceptions for invalid outputs.

The services exclusively comply with repository interfaces, adhering to the modular structure of the application.This ensures a clear separation of concerns and promotes the maintainability and scalability of the application.

The pattern is demonstrated in listing 7 which clearly shows the encapsulation of various operations related to parts management within the application. It interacts with repository interfaces such as `IPartRepository`, `IAssemblyRepository`, and `ISystemRepository` to perform CRUD (Create, Read, Update, Delete) operations on parts data.

6.2.3 Dependency Injection

Dependency injection is a software development pattern which enables to share dependencies for other components without the need for their infile inicializa-

```
public interface IAssemblyRepository
{
    Task <IEnumerable<CostAssembly?>> GetAssemblies();
    Task <IEnumerable<CostAssembly?>> GetAssembliesBySystemId(int
    ↪  systemId);
    Task <CostAssembly?> GetAssemblyById(int id);
    Task CreateAssembly(CostAssembly assembly);
    Task UpdateAssembly(CostAssembly assembly);
    Task DeleteAssembly(CostAssembly assembly);
}

public class AssemblyRepositoryImpl : IAssemblyRepository
{
    private readonly IDbContextFactory<DataContext>
    ↪  _dbContextFactory;

    public AssembliesRepositoryImpl( IDbContextFactory<DataContext>
    ↪  dbContextFactory)
    { ... }

    public async Task<IEnumerable<CostAssembly?>>
    ↪  GetAssembliesBySystemId()
    { ... }

    public async Task<IEnumerable<CostAssembly?>>
    ↪  GetAssembliesBySystemId(int systemId)
    { ... }

    public async Task<CostAssembly?> GetAssemblyById(int id)
    { ... }

    public async Task CreateAssembly(CostAssembly assembly)
    { ... }

    public async Task UpdateAssembly( CostAssembly assembly)
    { ... }

    public async Task DeleteAssembly(CostAssembly assembly)
    { ... }
}
```

Listing 6: Demonstration of repository pattern usage over the assembly entity

tion. For example to support the implemented layered architecture, rather then instancing `ISystemRepository` in the Assemblies Service, the dependancy injection using `@Inject` directive is used. It is used similarly throughout the whole implementation, not limited to the implemented components. Great example is the Radzen's `Dialog Service` which is part of Radzen Component library. All shown in the listing Listing 8, depiquting the assemblies table component.

```
public class PartsService : IPartsService
{
    private readonly IPartRepository _partsRepository;
    private readonly IAssemblyRepository _assemblyRepository;
    private readonly ISystemRepository _systemRepository;

    public PartsService(IPartRepository partsRepository,
    ↪  IAssemblyRepository assemblyRepository, ISystemRepository
    ↪  systemRepository)
    { ... }

    public async Task<IEnumerable<CostPart>> GetAllParts()
    { ... }

    public async Task<IEnumerable<CostPart?>>
    ↪  GetPartsByAssemblyId(int assemblyId)
    { ... }

    public async Task<CostPart?> GetPartById(int id)
    { ... }

    public async Task CreatePart(CostPart part)
    { ... }

    public async Task UpdatePart(CostPart part)
    { ... }

    public async Task DeactivatePart(int id)
    { ... }

    public async Task DeletePart(int id)
    {
        var partToDelete = await _partsRepository.GetPartById(id);
        if ( partToDelete is null)
            throw new
            ↪  KeyNotFoundException("Part ID  does not exist");
        await _partsRepository.DeletePart(partToDelete);
    }
}
```

Listing 7: Demonstration of service pattern usage over the parts entity

6.3 Database

A MySQL database was required for the application implementation. A local database was established using a Docker container to prevent corruption of the production database. The container setup is described in the listing Listing 9 representing the `docker-compose.yml` file.

```
@page "/assemblies/{SystemId:int}"
@using { ... }
@inject IAssembliesService AssembliesService
@inject DialogService DialogService

<PageTitle> Assemblies </PageTitle>

@if (_assemblies == null)
{  ... }
else
{
    <RadzenDataGrid ... > ... </RadzenDataGrid>
}

@code {
    [Parameter]
    public int SystemId { get; set; }

    private CostSystem? _system;
    IEnumerable<CostAssembly> _assemblies;
    RadzenDataGrid<CostAssembly> grid;

    async Task LoadAssemblies()
    {
        _assemblies = await
        ↪  AssembliesService.GetAssembliesOfSystem(SystemId);
    }

    ...

    async Task EditAssembly(int id)
    {
        await DialogService.OpenAsync<EditAssemblyDialog>( ... );
    }

}
```

Listing 8: Demonstration of he dependency injection usage

```
version: '3.8'

services:
  cost_db:
    image: mysql
    container_name: cm_db
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: cost
    ports:
      - "3307:3306"
```

Listing 9: Docker-compose.yml file containing the mock database setup 40

6.3.1 Database Connection

After successfully running the database container, the application connects to the database via a connection string saved in the `appsettings.json` file, as shown in Listing 10 . In the event of production deployment, the current database connection will be exchanged for the primary database of team eForce.

```
{
  ...,
  "ConnectionStrings":
  {
        "DefaultConnection":
            "Server=localhost;
             Database=cost;
             User=root;
             Password=root;
             Port=3307;
             Convert Zero Datetime=True;"
  },
  ...
}
```

Listing 10: Database connection string located in appsetings.json file

The connection string is loaded with other dependencies in the `Program.cs` file, which is the application's entry point, and subsequently establishes the database connection prior to the start of the application, as shown in Listing 11.

```
builder.Services.AddDbContextFactory<DataContext>(options =>
options.UseMySql(builder.Configuration
.GetConnectionString("DefaultConnection"), new
↪  MySqlServerVersion(new Version(8, 0, 28))));
```

Listing 11: Databse dependecies injected into the project

6.3.2 Database Model

As described in Chapter 5, the database was created using the code first approach using the Entity Framework. The resulting database schema is shown in Figure 6.1.

6.4 Authentification

The application utilizes an LDAP connection with a cookie authentication scheme. To facilitate implementation and testing, a test LDAP connection was used, similar to the approach taken with the mock database, to prevent any potential corruption to the active production service.

Despite initial attempts to develop a custom LDAP server for testing purposes, this proved to be inefficient and time-consuming, yielding minimal ben-

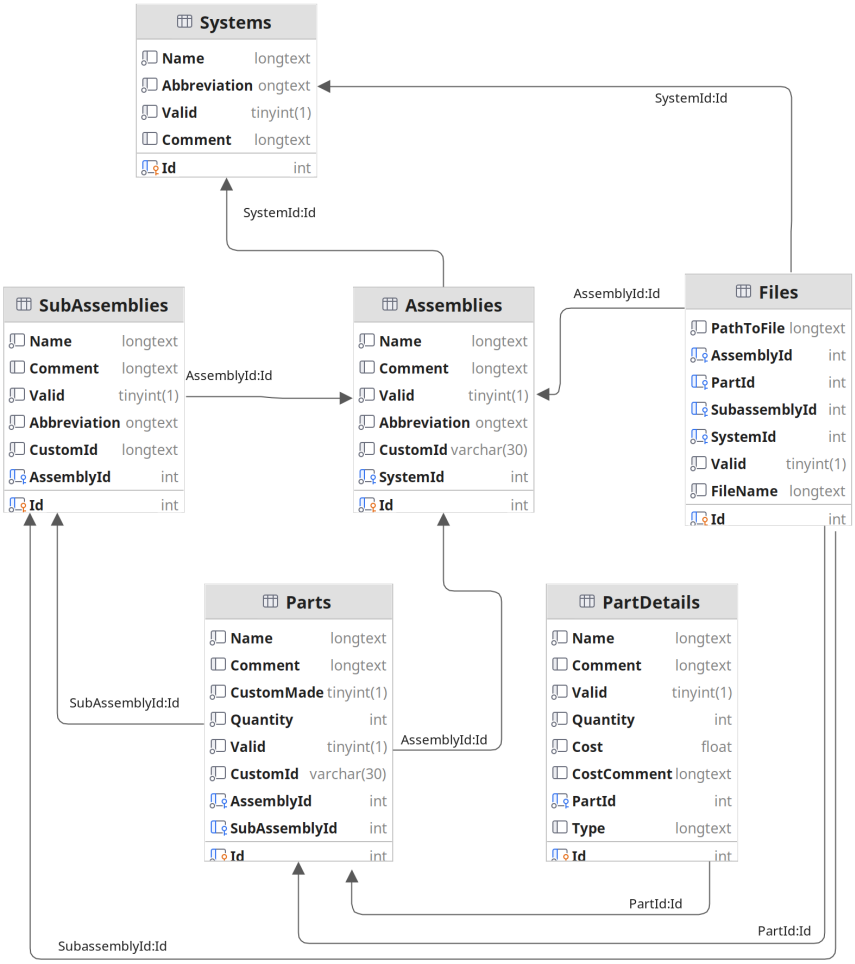


Figure 6.1: Implemented database model

efits for the implementation. Consequently, a test server provided by Forumsystems [45] was utilized instead. This, however, poses no problems for the application as the main goal is to allow only authenticated users to access the application's data.

The test server comprises two distinct user groups: mathematicians and scientists. Each user within these groups is named after renowned scientific figures like Euler or Newton. Uniformly, all users share the password 'password' to access the system.

6.4.1 LDAP Connection

The LDAP server configuration attributes are stored in the `appsettings.json` file, showcased in Listing 12. Authentication is managed by a dedicated login controller responsible for handling HTTP communication within the HTTP

context. Using controllers is the standardized approach for handling HTTP requests in the Blazor Server application, even though it is technically possible to handle them through services. The Login controller facilitates two HTTP requests: one for login and another for logout.

```
{
  ...
  "LDAP": {
    "Host": "ldap.forumsys.com",
    "BindDN": "cn=read-only-admin,dc=example,dc=com",
    "BindPassword": "password",
    "BaseDC": "dc=example,dc=com"
  },
  ...
}
```

Listing 12: LDAP connection attributes stored in `appsettings.json`

The login request displayed in Listing 13 establishes a connection to the LDAP server host on a predefined port. Subsequently, it binds the server domain name and authorization password, initializing the LDAP connection. Once established, the system searches for the username. If the credentials are invalid, an error is triggered. Otherwise a new identity claim is created. The user is then signed in using Cookie authentication and redirected to the application.

The logout function displayed in Listing 14, on the other hand, utilizes the HTTP context to access the current request. It clears the authentication cookies to ensure a successful logout.

```
[HttpGet("/account/logout")]
public async Task<IActionResult> Logout()
{
    await HttpContext.SignOutAsync(CookieAuthenticationDefaults
    .AuthenticationScheme);
    return LocalRedirect("/");
}
```

Listing 14: Logout function

6.4.2 Authentification in UI

To ensure that the authentication process is adequately reflected in the user interface, a `CascadingAuthenticationState` component is added to wrap the `Routing` component. This ensures the authentication state is available to all descendant components of the `Router` and is shown in Listing 15.

An `AuthorizedView` is used in the descendant components as shown in Listing 16, dividing the content into `Authorized` and `NotAuthorized` directives. This enables the presentation of different content based on the user's authentication state. If the application requires access to information about the signed

```
[HttpPost("/account/login")]
public async Task<IActionResult> Login(UserCredentials credentials)
{
   ...
    try
    {
        using (var connection = new LdapConnection())
        {
            connection.Connect(_ldapSettings.Host,
            ↪  LdapConnection.DefaultPort);
            connection.Bind(_ldapSettings.BindDN,
            ↪  _ldapSettings.BindPassword);

            var searchFilter = $"(uid={User.username})";
            var entities = connection.Search(_ldapSettings.BaseDC,
            ↪  LdapConnection.ScopeSub, searchFilter, null, false);

            ...

            connection.Bind(userDn, User.password);
            var claims = new[]
            {
                new Claim(ClaimTypes.Name, User.username),
            };

            var claimsIdentity = new ClaimsIdentity(claims,
            ↪  CookieAuthenticationDefaults.AuthenticationScheme);
            var claimsPrincipal = new ClaimsPrincipal(claimsIdentity);

            await HttpContext.SignInAsync(CookieAuthenticationDefaults
            .AuthenticationScheme, claimsPrincipal);
            return LocalRedirect("/systems");
        }
    }
    catch (LdapException e)
    {
        ...
    }
}
```

Listing 13: Login function responsible for user authentification

in user, it can retrieve them through `@context.User.Identity` command, as demonstrated in the navigation manager.

6.5 File Managment

A local storage directory was created for the implemented application. The directory path is specified in the `appsettings.json` file. Although the files are stored in the data storage, it is necessary to link the data information to corresponding entities such as `assemblies` or `parts`. Therefore, an entity for files was created, storing the original file name, all corresponding foreign keys, and, most importantly, the file path. The file path combines of the storage directory path and a randomly generated file name under which the file is stored. The original name is retained for download purposes, allowing the user to retrieve the file under its original file name.

```
<CascadingAuthenticationState>
    <Router AppAssembly="@typeof(App).Assembly">
        ...
    </Router>
</CascadingAuthenticationState>
```

Listing 15: Cascading authentification in App.razor file

```
<AuthorizeView>
    <Authorized>
        <div class="nav-item px-3">
            <h3>Hello @context.User.Identity.Name</h3>
        </div>
        <div class="nav-item px-3">
            <a href="/account/logout">Logout</a>
        </div>
    </Authorized>
    <NotAuthorized>
        ...
    </NotAuthorized>
</AuthorizeView>
```

Listing 16: Illustration of showcased content based on the user's authentification state

6.5.1 File Service

File management within the application utilizes a combination of repository and service, consistent with other entities in the application architecture. The `FileService` encompasses various functions, primarily for loading and downloading of files.

The `UploadFileAsync` function showcased in Listing 17 is responsible for saving the file into the storage under a generated file name as well as in the database. The `Download` function locates the file in the database, retrieves its file path, and generates a memory stream of the file.

Other functions within the `FileService` handle the generation of documents, such as the BOM and the Support Material file. These functions leverage the PDFSharp library [46] to transform data into PDFs. Both the `GenerateSupportFile` and `GenerateBOM` functions follow a similar structure. A custom header and footer are generated for each page, followed by the main content—either a PDF/image for the Support file or a data table for the BOM.

In order to generate a file, appropriate preparations are required, as shown in the Listing 18. Initially, the custom font is loaded from storage and then initialized. Afterward, a new PDF document is created, along with a new page. An XGraphics object, which acts as a drawing surface, is then associated with the page to facilitate the drawing of graphics onto the PDF page.

```
public async Task<bool> UploadFileAsync(IBrowserFile file, int
↪  maxFileSize, string[] allowedExtensions, CustomFile file_model)
{
    ...
    string newFileName =
    ↪  Path.ChangeExtension(Path.GetRandomFileName(),
    ↪  Path.GetExtension(file.Name));

    string path =
    ↪  Path.Combine(_configuration.GetValue<string>("FileStorage"),
    ↪  newFileName);

    string relativePath = Path.Combine(newFileName);

    await using FileStream fs = new(path, FileMode.Create);
    await file.OpenReadStream(maxFileSize).CopyToAsync(fs);

    file_model.PathToFile = relativePath;
    await _fileRepository.CreateFile(file_model);
    return true;

}
```

Listing 17: The file loading function

```
public async Task<Stream> GenerateSupportFile()
{
    if (PdfSharp.Fonts.GlobalFontSettings.FontResolver is null)
        {
            GlobalFontSettings.FontResolver = new
            ↪  CustomFontResolver();
        }
        XFont font = new XFont("Monserrat", 24);

        PdfDocument document = new PdfDocument();
        PdfPage page = document.AddPage();

        XGraphics gtx = XGraphics.FromPdfPage(page);

    ...
}
```

Listing 18: PDFSharp setup for file generation process

The `GenerateBOMHeader` function, presented in Listing 19, serves as a comprehensive demonstration of all PDFSharp functions employed throughout the document generation process. Each function within GenerateBOMHeader contributes to the construction of various document elements such as drawing lines, displaying images, rendering rectangles, and presenting strings.

```
public async Task GenerateBOMHeader(XGraphics gfx, string
↪  jpegSamplePath, string systemAbrr, PdfPage page, XFont font)
{
    gfx.DrawLine(XPens.Black, 20, 55, page.Width - 20, 55);

    XImage image = XImage.FromFile(jpegSamplePath);
    gfx.DrawImage(image, 10, 15, 90, 30);

    font = new XFont("Monserrat", 10);

    XBrush brush = await GetSystemColor(systemAbrr);
    gfx.DrawRectangle(brush, page.Width - 30, 15, 30, 25);

    gfx.DrawString(systemAbrr, font, XBrushes.Black, new XRect(-10,
    ↪  20, page.Width, page.Height), XStringFormats.TopRight );
}
```

Listing 19: `GenerateBomHeader` function

The snippets of the generated documents can be viewed in Figures B.8 and B.9.

6.6 Implemented Application

The screens of the implemented application are available to be viewed in the Appendix B

Chapter 7

Testing

Testing is considered an integral part of software development. It serves to measure the quality of software, uncover potential errors or bugs in the implementation, and determine the expected behavior of the software.

Tests can be categorized based on various aspects, such as classification based on inner structure [47]:

- **Black box** tests assess the functionalities of the application without prior knowledge of its implementation. Tests are provided only with inputs and expected outputs, thus typically focusing on fulfilling business logic requirements.
- **White box** tests have full access to the application's internal code and logic. They verify that the application's functions are operating correctly as intended. Additionally, they can facilitate testing beyond the main functionalities of the application.

Two categories of tests will be implemented in the application. The first category is unit tests, which are categorized as white box tests, as the developer has complete knowledge of the application's inner workings. The second type will be User acceptance tests, categorized as black box tests, and will be performed by members of the Eforce Formula Prague Team.

7.1 Unit Testing

Unit testing accesses the application's smallest parts, often individual methods or functions. It is conducted on code isolated from its dependencies and frequently involves mocking said dependencies. Unit tests should adhere to the FIRST principle [48], meaning they should be:

- Fast,
- Independent,
- Repeatable,
- Self-Validating,

- and Timely.

In ASP.NET Core applications, unit tests can be conducted using various tools, including XUnit. XUnit is a free, open-source tool for unit testing of C#, F#, VB.NET, and other .NET languages. [49]. It can be installed using the NuGet package manager directly from the IDE.

The test project is then set up in the same directory as the original application, and the tests are categorized into classes. It is best practice for the test project structure to mirror that of the original application, facilitating better navigation through the files.

As mentioned earlier, unit tests should be independent; thus, it is essential to mock all necessary dependencies. For this purpose, the Moq library is used. The following Listing 20 showcases the Moq library being utilized to simulate a repository dependency.

Each unit test is divided into three steps, commonly referred to as the three A's of unit testing [48]:

- **Arrange** – Prepare all necessary data for the tested method, including mocking dependencies.
- **Act** – Invoke the actual tested method.
- **Assert** – Assess if the method produced the anticipated result.

7.2 User Acceptance Testing

User acceptance testing represents the final phase of the application testing process before deployment to production. During this stage, the completed application is presented to the end users for whom it is designed. The users then perform a series of tests to validate the application's business logic implementation and provide valuable feedback and suggestions for potential improvements. [50]

7.2.1 Testers

To conduct the user acceptance testing, five members of Team eForce Prague Formula were invited to participate. These participants are university students without any IT background but have comprehensive knowledge in the Cost & Manufacturing discipline. The testers' ages range from 22 to 25 years old. None of the testers have prior experience in user acceptance testing.

7.2.2 Testing Scenerios

A testing scenario comprises a set of predetermined steps or actions executed to complete a specific functional process.

Each testing scenario provided to the testers includes a name describing the process, a starting point indicating the prerequisites completed before the scenario, and a series of steps to be followed. In total, 10 testing scenarios were executed, and their specifications can be found in Appendix C.

```
public class SystemsServiceTest
{
    [Fact]
    public async void GetAllSystemsAsync_ReturnsValidSystems()
    {
        // Arrange
        var validSystem1 = new CostSystem { Id = 1, Name =
        ↪  "Valid System 1", Valid = true };
        var invalidSystem = new CostSystem { Id = 2, Name =
        ↪  "Invalid System", Valid = false };
        var validSystem2 = new CostSystem { Id = 3, Name =
        ↪  "Valid System 2", Valid = true };

        var mockRepository = new Mock<ISystemRepository>();
        mockRepository.Setup(r => r.GetSystemsAsync())
            .ReturnsAsync(new List<CostSystem?> { validSystem1,
            ↪  invalidSystem, validSystem2 });

        var service = new SystemsService(mockRepository.Object);

        // Act
        var result = await service.GetAllSystemsAsync();

        // Assert
        Assert.NotNull(result);
        Assert.Equal(2, result.Count());
        Assert.Contains(validSystem1, result);
        Assert.Contains(validSystem2, result);
        Assert.DoesNotContain(invalidSystem, result);
    } ...
}
```

Listing 20: Example of unit test for `SystemService`

Printed copies of the testing scenarios were provided to the testers for execution. The tests were performed on the developer's computer, with the developer present to offer any necessary assistance.

7.2.3 Testing Evaulation

All testers completed the testing scenarios without encountering any system malfunctions. They expressed overall satisfaction with the application design and found the testing scenarios straightforward to follow. Additionally, they appreciated the option to edit the Make/Buy attribute in the part entity, as it is not supported in the FSG Portal and it was very common to acidentally chcek the wrong box and without the ability for correction, the part had to be deleted and repeatably created. The implemented application solves said issue and therefore improves the overall usability of the application.

The testers, however, proposed various improvements to be made. Three out of the five testers noted an issue during testing scenarios 8 and 9, where the file generation process took a few minutes between clicking the button and

the automated download of the file. This delay led to multiple clicks on the button and subsequent file downloads, which was not the desired behavior. To address this issue, an alert was implemented after testing to inform users that the documents were being generated and prompted them to wait patiently.

Another suggestion from the testers was to allow multiple file uploads. One tester accidentally selected multiple files in the testing scenerio 3, triggering an error message indicating that only one file can be uploaded. Although this was not the intended scenario, the tester suggested that allowing multiple file uploads could be beneficial. Therefore, the file upload function was updated after testing to support this functionality.

Lastly, one tester proposed supporting the reordering of table contents, as the resulting BOM document displays data in the order it was inputted into the system. Although this was identified as a Could requirement in the Chapter 3, it was not implemented in the application due to insufficient time availability. Consequently, the request was acknowledged and marked as a high priority for future development.

Chapter 8

Evaluation

The concluding chapter of this thesis evaluates the achieved results and outlines possibilities for future development.

8.0.1 Evaulation of Results

The analysis focused on the Formula Student competition, specifically looking at the Cost & Manufacturing discipline and the approach taken by team eForce Prague Formula. This analysis led to the definition of both functional and non-functional requirements, along with a set of use cases outlining the expected application functionalities.

The following chapter researched the essential technologies for the application, including hosting models, relational database creation, and authentication processes, along with the reasonings behind the implementation decisions. Additionally, the chapter presented the UI design, which served as a blueprint for the implemented application.

In the practical phase, a functional prototype of the application was developed, following the standard sowtfare engineering practices. Comprehensive unit testing and user acceptance testing by eForce Prague Formula team members confirmed the alignment of business logic and identified opportunities for future development.

In the last step of the implementation, the application was containerized and prepared for future deployment. However, it is not currently deployed to production, as the eForce Prague Formula team is undergoing restructuring of their IT infrastructure to enhance server security.

While the application successfully meets all non-functional requirements as a containerized web application built on the ASP.NET Core framework with functioning authentication and MySQL database, only *Must have* and *Should Have* functional requirements were fully implemented due to time constraints.

8.0.2 Future Development

The top priority for future development is the implementation of the remaining functional requirements, with drag and drop stamping being particularly critical, as was highlighted during user acceptance testing.

Another possible area for future development is the implementation of an admin portal. This portal would differentiate users into administrators and regular users, granting administrators higher authority. Administrators would be able to create new systems, which would prove valuable in adapting to any changes in competition rules. Additionally, they could verify inputted information, marking it as verified and correct. This feature would not only provide the team with a clearer overview of their data collection status but also enhance the overall quality of resulting documents.

Conclusion

The main goal of this thesis was to develop a web application in collaboration with the eForce Prague Formula Team that supports efficient data collection and required documentation production for the Cost & Manufacturing discipline in the Formula Student Competition. Several partial objectives have been fulfilled to achieve the primary goal.

Firstly, a comprehensive analysis of the Formula Student competition and the specific requirements of the Cost & Manufacturing event was conducted. This analysis enabled the identification and categorization of necessary information, as well as the identification of potential optimizations to enhance the team's approach to the discipline. These findings were used to define the functional and non-functional requirements with corresponding use cases and domain conceptual diagram.

The subsequent design phase highlighted .NET development possibilities and outlined chosen technologies and architecture, focusing on creating an intuitive user interface. The implementation chapter details the integration of third-party resources, including a relational database management system and authentication provider, while discussing development patterns.

Testing played a crucial role, with unit testing utilizing XUnit and Moq to assess services and user acceptance testing conducted by the Eforce Formula Prague Team, validating usability and business logic alignment.

Finally, in the concluding chapter, the results of the thesis were evaluated, and possibilities for future development were outlined. With that, all partial objectives of this thesis were concluded, and therefore, the assignment was successfully completed.

Bibliography

[1] Formula Student Germany GmbH. FSG: Concept. https://www.formulastudent.de/about/concept/, (Accessed on 04/01/2024).

[2] Institution of Mechanical Engineers. History of Formula Student. https://www.imeche.org/events/formula-student/about-formula-student/history-of-formula-student, (Accessed on 03/10/2024).

[3] Formula Student Germany GmbH. FSG: Disciplines. https://www.formulastudent.de/about/disciplines/, (Accessed on 03/10/2024).

[4] eForce Prague Formula. eForce_brozurka_2024_EN_.pdf - Nextcloud. https://eforce1.feld.cvut.cz/cloud/apps/files/?dir=/EFORCE_PRG/!2024%20-%20CTU.24/GRAFIKA/Velk%C3%A1%20bro%C5%BEura&openfile=1069404, 2023, (Accessed on 03/11/2024).

[5] Banner GmbH. Energy recuperation - What is it? https://www.bannerbatterien.com/en/Battery-knowledge/3-Banner-Lexicon-Energy-Recuperation, (Accessed on 03/10/2024).

[6] Formula Student Germany GmbH. WRL - Formula Student Electric. https://fs-world.org/E/261/, (Accessed on 03/15/2024).

[7] Formula Student Germany GmbH. Formula Student Rules 2024, Version: 1.1, Rev-aef3d92. https://www.formulastudent.de/fileadmin/user_upload/all/2024/rules/FS-Rules_2024_v1.1.pdf, 2023, (Accessed on 03/11/2024).

[8] Formula Student Germany GmbH. FSG: Login. https://www.formulastudent.de/login, (Accessed on 04/25/2024).

[9] Gilmore, S. CS2Ah0405-SoftwareRequirements.pdf. https://www.inf.ed.ac.uk/teaching/courses/ip/CS2Ah0405-SoftwareRequirements.pdf, 2003, (Accessed on 03/28/2024).

[10] AltexSoft. Functional and Nonfunctional Requirements Specification. https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/, (Accessed on 04/25/2024).

[11] Brush, K. What is the MoSCoW Method? https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method, 2023, (Accessed on 04/02/2024).

[12] IBM Corporation. Use-case diagrams. https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case, 2021, (Accessed on 03/28/2024).

[13] Visual Paradigm. Types of Actor in a Use Case Model. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/types-of-actor-in-use-case-model/, (Accessed on 04/25/2024).

[14] JGraph Ltd. draw.io. https://www.drawio.com/, (Accessed on 04/25/2024).

[15] Seidl, M.; Scholz, M.; et al. *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Undergraduate Topics in Computer Science, Springer International Publishing, 2015, ISBN 9783319127415. Available from: https://books.google.cz/books?id=ggLsoQEACAAJ

[16] Codes, C. A Brief History of .NET (dotnet). Whether you're new to the world of .NET. https://medium.com/calvin-codes/a-brief-history-of-net-ec4c14adf441, 2023, (Accessed on 04/21/2024).

[17] Townsend, J. The History of .NET. https://omnitech-inc.com/blog/the-history-of-net/, 2022, (Accessed on 04/21/2024).

[18] Prashant. What Is Blazor and How Does It Work? https://programmers.io/blog/what-is-blazor-and-how-does-it-work/, 2023, (Accessed on 04/21/2024).

[19] Microsoft Corporation. Overview of ASP.NET Core SignalR. https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-8.0, 2023, (Accessed on 04/21/2024).

[20] Lucio, V. G. G. 3 Different Hosting Models in Blazor. https://www.syncfusion.com/blogs/post/3-blazor-hosting-models, 2023, (Accessed on 04/21/2024).

[21] Mohanty, A. Blazor Server vs. Blazor WebAssembly: Pros and cons of each approach. https://www.c-sharpcorner.com/article/blazor-server-vs-blazor-webassembly-pros-and-cons-of-each-approach/, (Accessed on 04/23/2024).

[22] Microsoft Corporation. ASP.NET Core Blazor hosting models. https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-8.0, (Accessed on 04/23/2024).

[23] Price, M. *C# 10 and . NET 6 - Modern Cross-Platform Development - Sixth Edition: Build Apps, Websites, and Services with ASP. NET Core 6, Blazor, and EF Core 6 Using Visual Studio 2022 and Visual Studio Code*. Expert insight, Packt Publishing, 2021, ISBN 9781801077361. Available from: https://books.google.cz/books?id=8lS5zgEACAAJ

[24] EntityFrameworkTutorial.net. Database First development with Entity Framework. https://www.entityframeworktutorial.net/entityframework6/choosing-development-approach-with-entity-framework.aspx, (Accessed on 04/23/2024).

[25] Microsoft Corporation. Architectural principles - .NET. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles, 2023, (Accessed on 05/16/2024).

[26] Plutora. What Is Compliance Management and How Do You Get Started? https://www.plutora.com/blog/understanding-the-dry-dont-repeat-yourself-principle, 2023, (Accessed on 05/16/2024).

[27] Sheldon, R. What is encapsulation (object-orientated programming)? https://www.techtarget.com/searchnetworking/definition/encapsulation, 2023, (Accessed on 05/16/2024).

[28] Cambi, L. System Design: Dependency Inversion Principle. https://www.baeldung.com/cs/dip, 2024, (Accessed on 05/16/2024).

[29] John, G. Four Must-Know Software Architectural Patterns for Every Developer: A brief overview. https://www.linkedin.com/pulse/four-must-know-software-architectural-patterns-every-developer-john-gvobf, (Accessed on 04/23/2024).

[30] miniOrange Security Software Pvt Ltd. What is Authentication? Different Types of Authentication. https://www.miniorange.com/blog/different-types-of-authentication-methods-for-security/, (Accessed on 05/07/2024).

[31] Choi, K. Stateful and stateless authentication. https://medium.com/@kennch/stateful-and-stateless-authentication-10aa3e3d4986, (Accessed on 05/07/2024).

[32] Microsoft Corporation. Upload files in ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads?view=aspnetcore-6.0, (Accessed on 05/02/2024).

[33] HubSpot, Inc. User Interface (UI) Design: What Is It? The Beginner's Guide. https://blog.hubspot.com/website/ui-design, (Accessed on 04/30/2024).

[34] Radzen. Free Blazor Components. https://blazor.radzen.com/, (Accessed on 05/07/2024).

[35] Megabit. Blazorise - Blazor Component Library. https://blazorise.com/, (Accessed on 05/07/2024).

[36] MudBlazor. MudBlazor - Blazor Component Library. https://mudblazor.com/, (Accessed on 05/07/2024).

[37] Figma, Inc. Figma: The Collaborative Interface Design Tool. https://www.figma.com/, (Accessed on 04/23/2024).

[38] Wrike, Inc. What Is Software Project Management? https://www.wrike.com/project-management-guide/faq/what-is-software-project-management/, (Accessed on 05/13/2024).

[39] Dremio. Data Versioning. https://www.dremio.com/wiki/data-versioning/, (Accessed on 05/13/2024).

[40] TechTarget. What is GitLab? https://www.techtarget.com/whatis/definition/GitLab, (Accessed on 05/13/2024).

[41] International Business Machines Corporation. What Is Containerization? https://www.ibm.com/topics/containerization, (Accessed on 05/13/2024).

[42] Amazon Web Services, Inc. What is Docker? https://aws.amazon.com/docker/, (Accessed on 05/13/2024).

[43] GitLab Inc. What is CI/CD? https://about.gitlab.com/topics/ci-cd/, (Accessed on 05/13/2024).

[44] GitLab Inc. Get started with GitLab CI/CD. https://docs.gitlab.com/ee/ci/, (Accessed on 05/13/2024).

[45] Forum Systems. Online LDAP Test Server - Forum Systems. https://www.forumsys.com/2022/05/10/online-ldap-test-server/, (Accessed on 05/12/2024).

[46] empira Software GmbH. Home of PDFsharp and MigraDoc Foundation - PDFsharp & MigraDoc. https://www.pdfsharp.net/, (Accessed on 05/13/2024).

[47] Thomas, A. Software Application Testing | What it is, Types & How to do? https://testsigma.com/blog/software-application-testing/, 2023, (Accessed on 04/19/2024).

[48] Rahman, T. F.I.R.S.T principles of testing. First principles of testing stand for. https://medium.com/@tasdikrahman/f-i-r-s-t-principles-of-testing-1a497acda8d6, 2019, (Accessed on 04/19/2024).

[49] .NET Foundation. Home > xUnit.net. https://xunit.net/, (Accessed on 04/19/2024).

[50] Hamilton, T. What is User Acceptance Testing (UAT)? Examples. https://www.guru99.com/user-acceptance-testing.html, (Accessed on 05/14/2024).

Acronyms

BOM Bill of Materials. 7–10, 45

CBOM Costed Bill of Materials. 7, 9, 10

CD Continuous Delivery. 33–35

CI Continuous Integration. 33–35

CRD Cost Report Documents. 7

CV Combustion engine vehicle. 3

DBOM Detailed Bill of Materials. 7–10

DV Driverless vehicle. 3

EV Electrical vehicle. 3

UI User Interface. 28

Appendix A

Formula Student Specifications

System
Brake System
Electrical
Engine & Drivetrain
Chassis & Body
Miscellaneous, Fit & Finish & Assembly
Steering System
Suspension System
Wheels, Wheel Bearings & Tires

Table A.1: List of systems

System	Assembly Name
Brake system	Balance Bar Brake Discs Brake Fluid Brake Lines Brake Master Cylinder Brake Pads Brake System Front Brake System Rear Calipers Fasteners Proportioning Valve Other
Electrical	Accumulator Accumulator Container Accumulator Cooling Autonomous Acceleration System Autonomous Box Autonomous Brake System Autonomous Camera System Autonomous EBS

System	Assembly Name
	Autonomous Lidar System
	Autonomous Radar System
	Autonomous Steering System
	Autonomous System
	BSPD
	Brake Light
	Bulbs
	Control Unit
	Dash Panel
	ECM/Engine Electronics
	Fuses
	HV-Batery
	Indicator Lights
	Kill Switch
	LV-Battery
	Oil Pressure Gage/Light
	Relays
	Sensors
	Solenoids
	Starter Button
	TSAL/ASSI
	Tachometer
	Telemetry
	VCU
	Water Temperature Gage
	Wire Harness/Connectors
Engine & Drivetrain	Air Filter
	Airbox
	Axles
	CV Joints / U Joints
	Carburetor
	Chain / Belt
	Clutch
	Coolant
	Coolant Lines
	Cooling System
	Differential
	Differential Bearings
	Differential Mounts
	Driveshaft Assembly
	Drivetrain Assembly
	Engine
	Engine Mounts
	Engine/Diff Oil
	Exhaust Manifold
	Exhaust System
	Fuel Pressure Reg.
	Fuel Pump
	Fuel Tank - NOT THE HV-Battery

System	Assembly Name
	Fuel Vent/Check Valve
	Gearbox
	Hose Clamps
	Intake Manifold
	Intake System
	Muffler
	Oil Cooler
	Overflow Bottles
	Restrictor
	Shields
	Sprocket/Pulleys
	Throttle Body
	Transmission
	Turbo/Super Charger
	Other
Suspension System	A-Arms front lower
	A-Arms front upper
	A-Arms rear lower
	A-Arms rear upper
	Anti Roll Bar Front
	Anti Roll Bar Rear
	Bell Cranks
	Bell Cranks Front
	Bell Cranks Rear
	Brackets
	Front A/Arms or Equivalent
	Front Uprights
	Push/Pullrod Front
	Push/Pullrod Rear
	Pushrods/Pullrods
	Rear A/Arms or Equivalent
	Rear Uprights
	Rod Ends
	Shocks Front
	Shocks Rear
	Springs
	Suspension Mechanism
	Tie Rod - Front
	Tie Rod - Rear
	Other
	Aerodynamic Wing (if used)
	Aerodynamics DRS
	Aerodynamics Front Wing
	Aerodynamics Rear Wing
	Aerodynamics Side Wings
	Aerodynamics Underbody
	Autonomous EBS
	Body Attachments

System	Assembly Name
Chassis & Body	Autonomous EBS Body Attachments Body Material Body Processing Brackets Chassis Assembly Clutch Crash Box Floor Pan Frame / Frame Tubes Front Hoop Impact Attenuator Main Hoop Monocoque Mounts Integral to Frame Pedal (Accelerator) Pedal (Brake) Shifter Shifter Cable/Linkage Throttle Controls Tube End Preps Tubes Cuts/Bends Other
Miscellaneous, Fit & Finish & Assembly	Driver's Harness Fire Wall Headrest / Restraints Mirrors Paint - Body Paint - Frame Seats Shields Other
Steering System	Steering Rack Steering Shaft Steering Wheel Steering Wheel Quick Release Tie Rods Other
Wheels, Wheel Bearings & Tires	Front Hubs Lug Nuts Rear Hubs Tires Valve Stems Wheel Bearings Wheel Studs Wheels Other

Table A.2: List of assemblies names predefined by the rules

FORMULA STUDENT

CBOM - TU Prague (FS CZ, TID394, Czech Republic) 2023-07-07 05:55 CEST

BR (Brake System)	**Balance Bar**				250,71€	394-BR-A0003
[Assembly Processes]		Make			1 x 25,00€ = 25,00€	394-BR-A0003P
Process Assemble	*attaching BB parts to master cylinders*		1 x 10,00€ = 10,00€	*20m*		
Process Assemble	*attaching servo motor*		1 x 5,00€ = 5,00€	*10m*		
Process Assemble	*assembling brake ballance mechanism*		1 x 10,00€ = 10,00€	*20m*		
Join pin	*CuSn8*	Make			2 x 29,83€ = 59,66€	394-BR-00007
Material Bronze	*CuSn8*		1 x 2,00€ = 2,00€	*Price per piece*		
Process Machining setup	*attaching material & uploading programe*		1 x 2,50€ = 2,50€	*5m*		
Process Machining	*Lathe*		1 x 6,83€ = 6,83€	*4m*		
Process Machining setup	*reattaching to milling machine*		1 x 2,50€ = 2,50€	*5m*		
Process Machining	*Milling machining*		1 x 13,50€ = 13,50€	*5m*		
Process Other Quality check	*checking given dimensions and tolerances*		1 x 2,50€ = 2,50€	*5m*		
Balance bar connection	*7075 T6*	Make			2 x 36,00€ = 72,00€	394-BR-00008
Material Aluminum	*7075 T6*		1 x 1,50€ = 1,50€	*price per piece*		
Process Machining setup	*attaching material & uploading programe*		1 x 2,50€ = 2,50€	*5m*		
Process Machining	*Milling machining*		1 x 13,50€ = 13,50€	*5m*		
Process Machining setup	*Reattaching material*		1 x 2,50€ = 2,50€	*5m*		
Process Machining	*Milling machining*		1 x 13,50€ = 13,50€	*5m*		
Process Other Quality check	*Checking tolerances and dimensions*		1 x 2,50€ = 2,50€	*5m*		
Bearing	*SKF - GE12C*	Buy			1 x 20,00€ = 20,00€	394-BR-00009
Material Bought Part	*n/a*		1 x 20,00€ = 20,00€	*price per piece*		
Brake Balance bar	*42CrMo4*	Make			1 x 52,60€ = 52,60€	394-BR-00010
Material Steel	*42CrMo4*		1 x 8,00€ = 8,00€	*price per piece*		
Process Machining setup	*attaching material @ uploading programe*		1 x 2,50€ = 2,50€	*5m*		
Process Machining	*lathe machining*		1 x 17,09€ = 17,09€	*10m*		
Process Machining setup	*Reattaching material*		1 x 2,50€ = 2,50€	*5m*		
Process Machining	*lathe machining*		1 x 13,67€ = 13,67€	*8m*		
Process Other Quality check	*checking tolerances, dimensions, threads*		1 x 5,00€ = 5,00€	*10m*		
Process Machining setup	*attaching material @ uploading programe*		1 x 2,50€ = 2,50€	*5m*		
Process Machining	*milling machining*		1 x 1,35€ = 1,35€	*1m*		
Connector	*Meritor 720.64*	Buy			1 x 4,00€ = 4,00€	394-BR-00011
Material Bought Part	*n/a*		1 x 4,00€ = 4,00€	*price per piece*		

Figure A.1: Partial CBOM submitted by team eForce in Formula Student Czech Re- public 2023

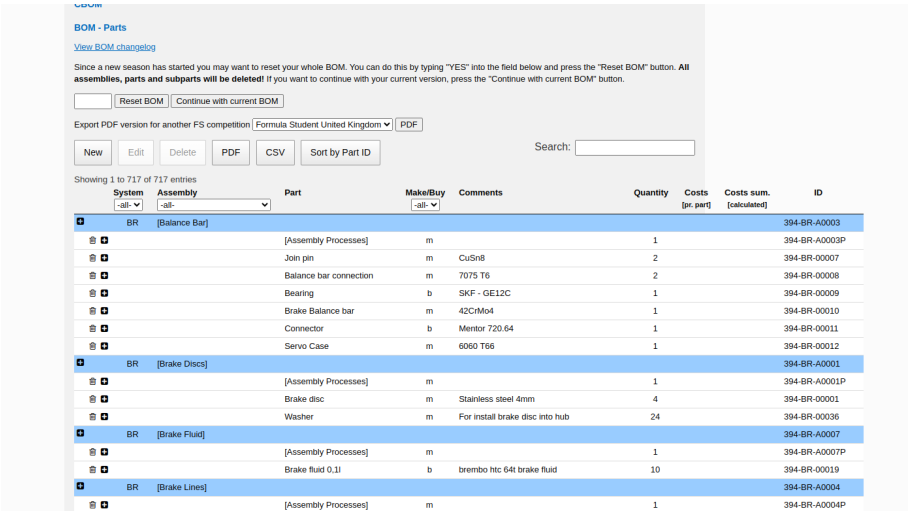


Figure A.2: FSG Portal

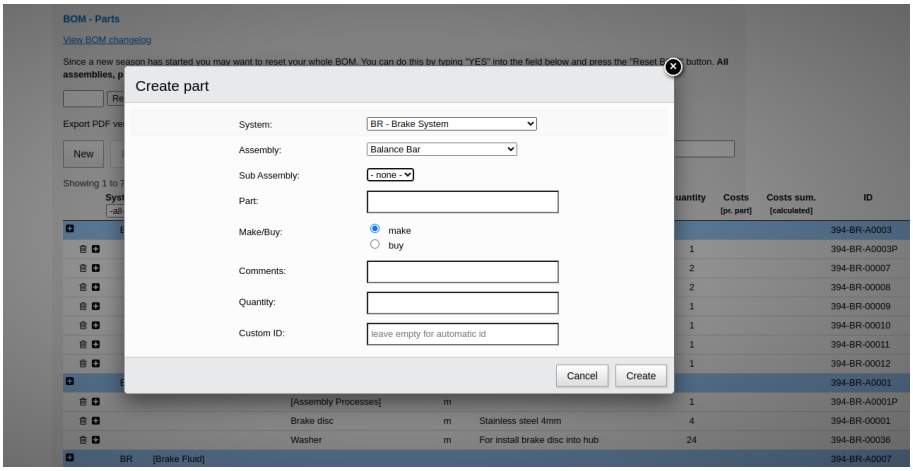
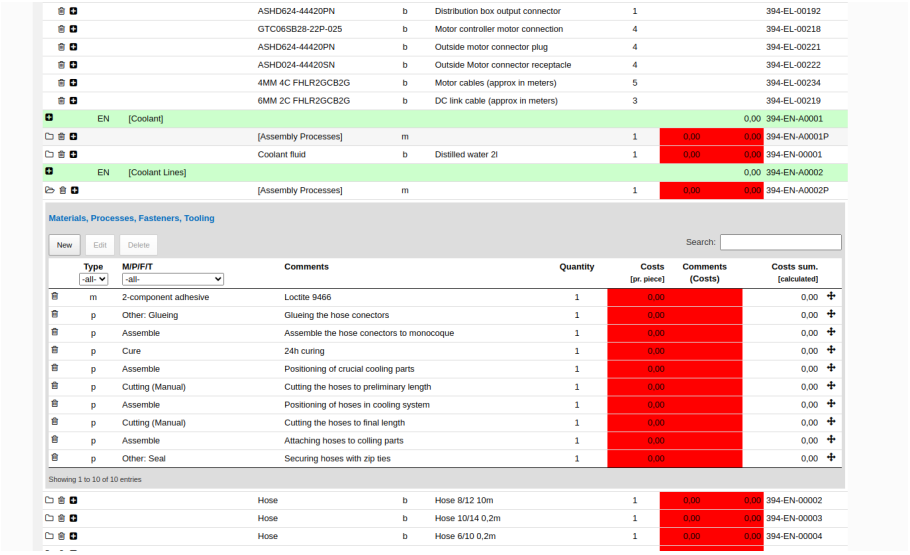


Figure A.3: Creating Part through FSG Portal.



		ASHD624-44420PN	b	Distribution box output connector	1			394-EL-00192
		GTC06SB28-22P-025	b	Motor controller motor connection	4			394-EL-00218
		ASHD624-44420PN	b	Outside motor connector plug	4			394-EL-00221
		ASHD024-44420SN	b	Outside Motor connector receptacle	4			394-EL-00222
		4MM 4C FHLR2GCB2G	b	Motor cables (approx in meters)	5			394-EL-00234
		6MM 2C FHLR2GCB2G	b	DC link cable (approx in meters)	3			394-EL-00219
	EN	[Coolant]					0,00	394-EN-A0001
		[Assembly Processes]	m		1	0,00	0,00	394-EN-A0001P
		Coolant fluid	b	Distilled water 2l	1	0,00	0,00	394-EN-00001
	EN	[Coolant Lines]					0,00	394-EN-A0002
		[Assembly Processes]	m		1	0,00	0,00	394-EN-A0002P

Materials, Processes, Fasteners, Tooling

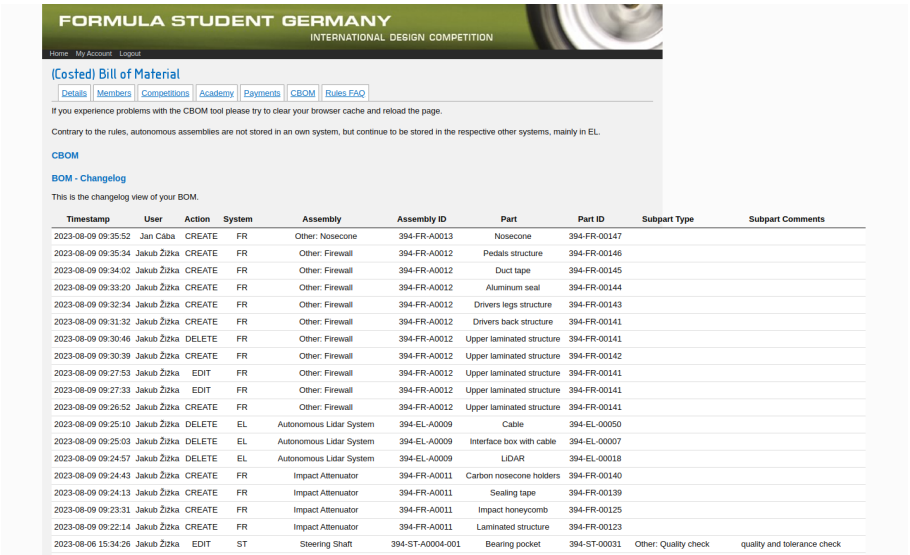
New | Edit | Delete | Search:

Type	M/P/F/T	Comments	Quantity	Costs [pr. piece]	Comments (Costs)	Costs sum. [calculated]
	-all-	-all-				
m	2-component adhesive	Loctite 9466	1	0,00		0,00
p	Other: Glueing	Glueing the hose conectors	1	0,00		0,00
p	Assemble	Assemble the hose conectors to monocoque	1	0,00		0,00
p	Cure	24h curing	1	0,00		0,00
p	Assemble	Positioning of crucial cooling parts	1	0,00		0,00
p	Cutting (Manual)	Cutting the hoses to preliminary length	1	0,00		0,00
p	Assemble	Positioning of hoses in cooling system	1	0,00		0,00
p	Cutting (Manual)	Cutting the hoses to final length	1	0,00		0,00
p	Assemble	Attaching hoses to colling parts	1	0,00		0,00
p	Other: Seal	Securing hoses with zip ties	1	0,00		0,00

Showing 1 to 10 of 10 entries

		Hose	b	Hose 8/12 10m	1	0,00	0,00	394-EN-00002
		Hose	b	Hose 10/14 0,2m	1	0,00	0,00	394-EN-00003
		Hose	b	Hose 6/10 0,2m	1	0,00	0,00	394-EN-00004

Figure A.4: CBOM on FSG Portal



FORMULA STUDENT GERMANY
INTERNATIONAL DESIGN COMPETITION

Home My Account Logout

(Costed) Bill of Material

Details | Members | Competitions | Academy | Payments | CBOM | Rules FAQ

If you experience problems with the CBOM tool please try to clear your browser cache and reload the page.

Contrary to the rules, autonomous assemblies are not stored in an own system, but continue to be stored in the respective other systems, mainly in EL.

CBOM

BOM - Changelog

This is the changelog view of your BOM.

Timestamp	User	Action	System	Assembly	Assembly ID	Part	Part ID	Subpart Type	Subpart Comments
2023-08-09 09:35:52	Jan Cába	CREATE	FR	Other: Nosecone	394-FR-A0013	Nosecone	394-FR-00147		
2023-08-09 09:35:34	Jakub Žižka	CREATE	FR	Other: Firewall	394-FR-A0012	Pedals structure	394-FR-00146		
2023-08-09 09:34:02	Jakub Žižka	CREATE	FR	Other: Firewall	394-FR-A0012	Duct tape	394-FR-00145		
2023-08-09 09:33:20	Jakub Žižka	CREATE	FR	Other: Firewall	394-FR-A0012	Aluminum seal	394-FR-00144		
2023-08-09 09:32:34	Jakub Žižka	CREATE	FR	Other: Firewall	394-FR-A0012	Drivers legs structure	394-FR-00143		
2023-08-09 09:31:32	Jakub Žižka	CREATE	FR	Other: Firewall	394-FR-A0012	Drivers back structure	394-FR-00141		
2023-08-09 09:30:46	Jakub Žižka	DELETE	FR	Other: Firewall	394-FR-A0012	Upper laminated structure	394-FR-00141		
2023-08-09 09:30:39	Jakub Žižka	CREATE	FR	Other: Firewall	394-FR-A0012	Upper laminated structure	394-FR-00142		
2023-08-09 09:27:53	Jakub Žižka	EDIT	FR	Other: Firewall	394-FR-A0012	Upper laminated structure	394-FR-00141		
2023-08-09 09:27:33	Jakub Žižka	EDIT	FR	Other: Firewall	394-FR-A0012	Upper laminated structure	394-FR-00141		
2023-08-09 09:26:52	Jakub Žižka	CREATE	FR	Other: Firewall	394-FR-A0012	Upper laminated structure	394-FR-00141		
2023-08-09 09:25:10	Jakub Žižka	DELETE	EL	Autonomous Lidar System	394-EL-A0009	Cable	394-EL-00050		
2023-08-09 09:25:03	Jakub Žižka	DELETE	EL	Autonomous Lidar System	394-EL-A0009	Interface box with cable	394-EL-00007		
2023-08-09 09:24:57	Jakub Žižka	DELETE	EL	Autonomous Lidar System	394-EL-A0009	LiDAR	394-EL-00018		
2023-08-09 09:24:43	Jakub Žižka	CREATE	FR	Impact Attenuator	394-FR-A0011	Carbon nosecone holders	394-FR-00140		
2023-08-09 09:24:13	Jakub Žižka	CREATE	FR	Impact Attenuator	394-FR-A0011	Sealing tape	394-FR-00139		
2023-08-09 09:23:31	Jakub Žižka	CREATE	FR	Impact Attenuator	394-FR-A0011	Impact honeycomb	394-FR-00125		
2023-08-09 09:22:14	Jakub Žižka	CREATE	FR	Impact Attenuator	394-FR-A0011	Laminated structure	394-FR-00123		
2023-08-06 15:34:26	Jakub Žižka	EDIT	ST	Steering Shaft	394-ST-A0004-001	Bearing pocket	394-ST-00031	Other: Quality check	quality and tolerance check

Figure A.5: Changelog on FSG Portal

Appendix B

Implementation

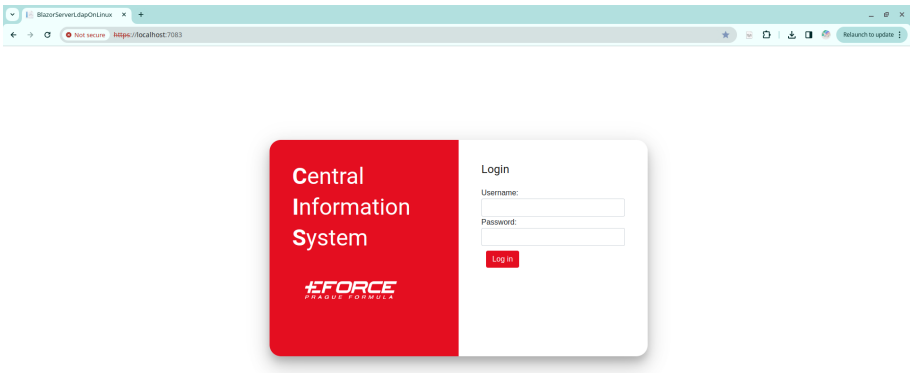


Figure B.1: Implemented Login page

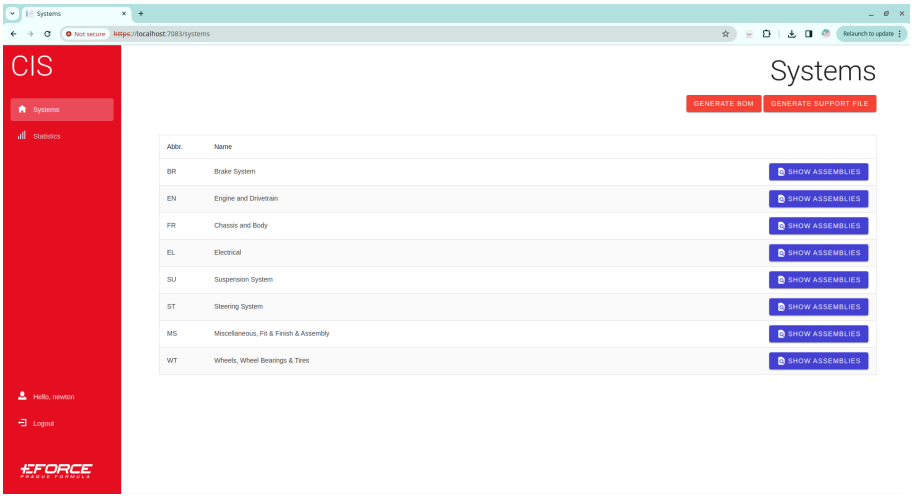


Figure B.2: Implemented Systems page

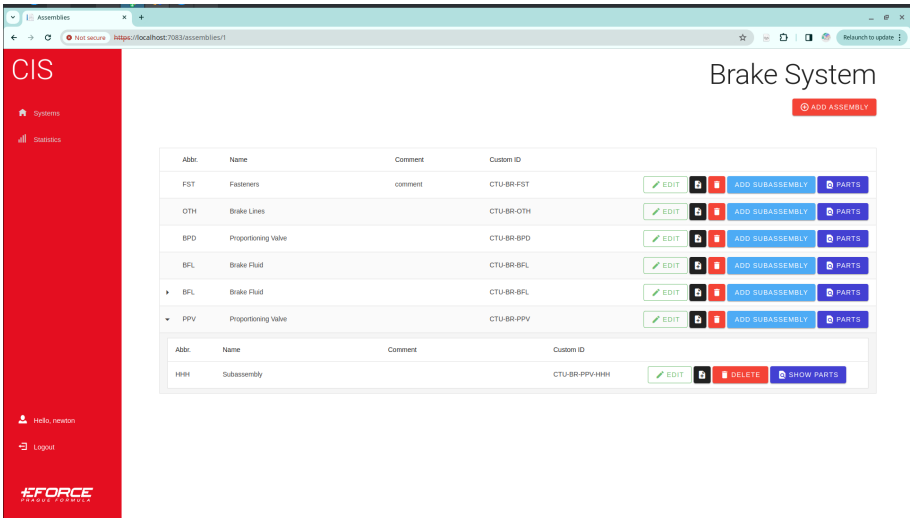


Figure B.3: Implemented Assemblies and subAssemblies page page

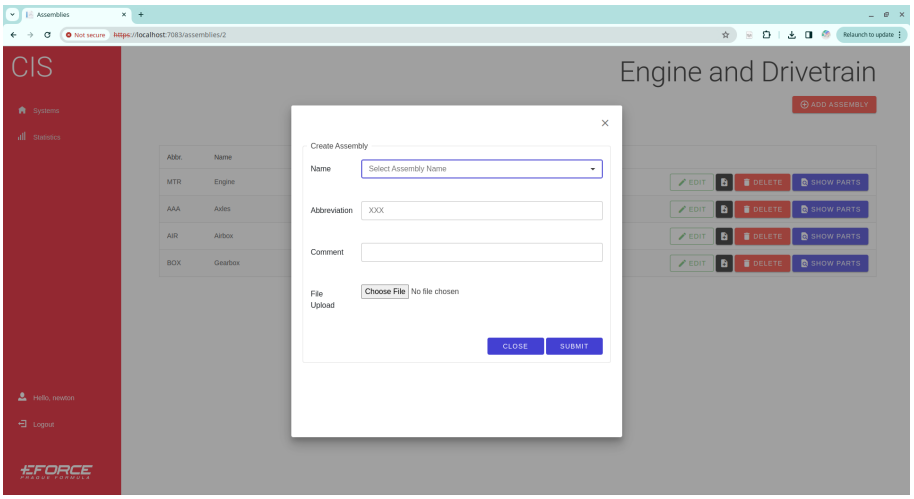


Figure B.4: Implemented dialog for creation of new assembly

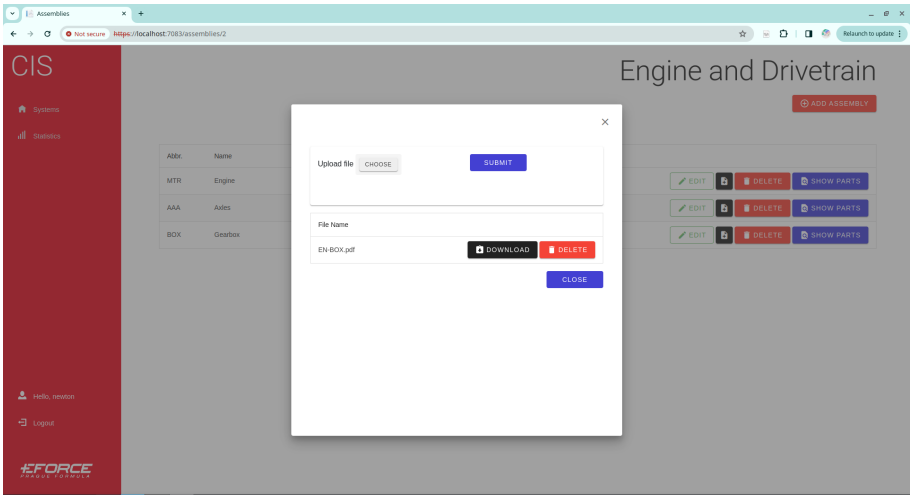


Figure B.5: Implemented file manager dialog window

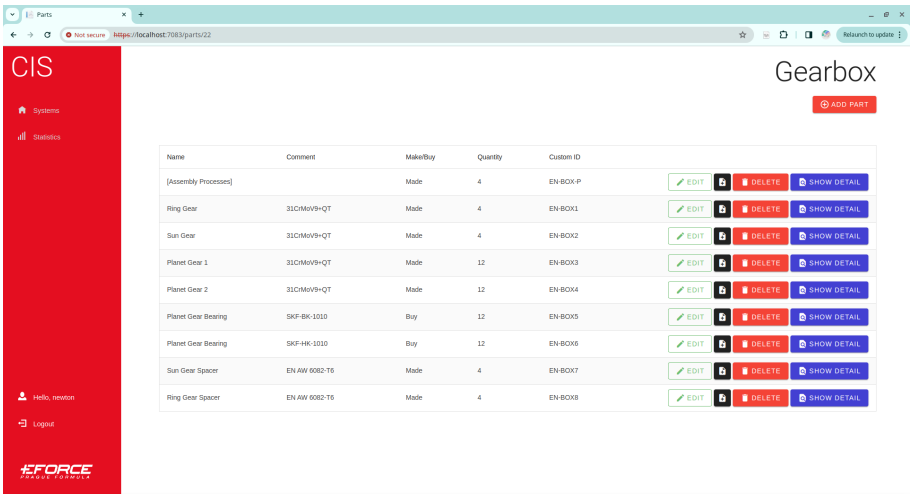


Figure B.6: Implemented page displaying parts of an assembly

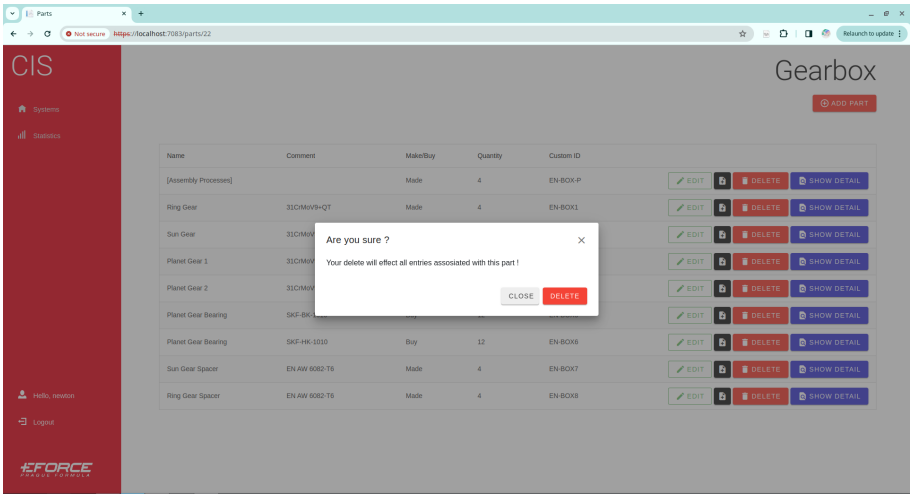


Figure B.7: Delete part confirmation dialog

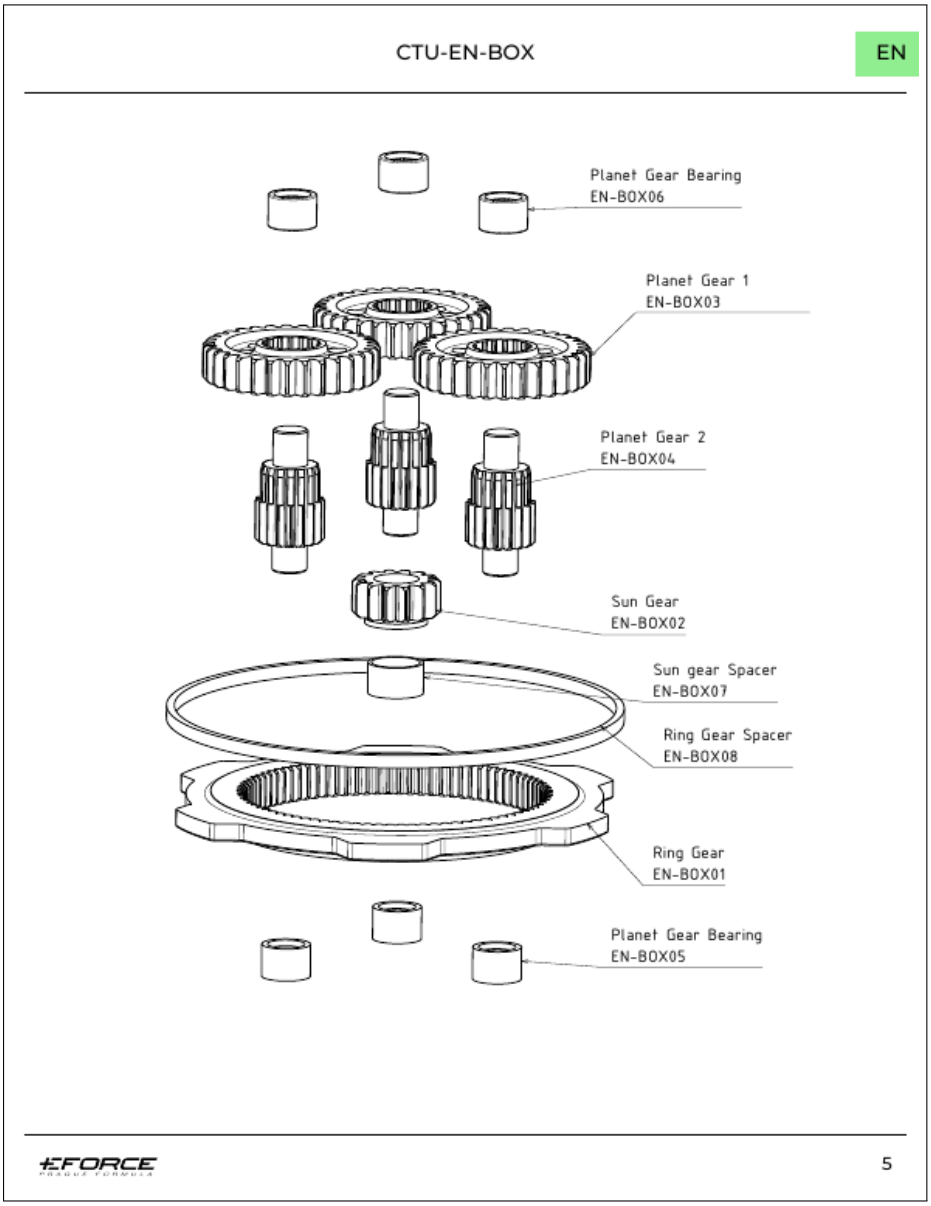


Figure B.8: A snippet of the Support material document generated using PFD-Sharp library

EN

EN (Engine and Drivetrain) Gearbox CTU-EN-BOX

[Assembly Processes]		Make	4x0.4=1.6€	EN-BOX-P
process	Preparing		1x0.01=0.01€	
fastener	Bolt		1x0.38=0.38€	
process	Hand Finish		1x0.01=0.01€	
Ring Gear	31CrMoV9-QT	Make	4x46.08=184.32€	EN-BOX1
material	Steel		1x5.62=5.62€	
process	Machining	31CrMoV9+QT DIN40-20 1st setup machining	1x25.17=25.17€	
process	EDM Wire	EDM wire machining	1x15.29=15.29€	
Sun Gear	31CrMoV9-QT	Make	4x0=0€	EN-BOX2
Planet Gear 1	31CrMoV9-QT	Make	12x0=0€	EN-BOX3
Planet Gear 2	31CrMoV9-QT	Make	12x0=0€	EN-BOX4
Planet Gear Bearing	SKF-BK-1010	Buy	12x0=0€	EN-BOX5
Planet Gear Bearing	SKF-HK-1010	Buy	12x0=0€	EN-BOX6
Sun Gear Spacer	EN AW 6082-T6	Make	4x0=0€	EN-BOX7
Ring Gear Spacer	EN AW 6082-T6	Make	4x0=0€	EN-BOX8

2024-05-13 12:07 +02:00

9

Figure B.9: A snippet of the BOM document generated using PFDSharp library

Appendix C

Testing Scenarios

C.1 Login

Starting Point

The web application is open on the *login* page, displaying the application name and a login form.

Steps

1. Input the string 'newton' into the username input field.
2. Input the password 'password' into the password input field.
3. Press the login button located under the login form fields.
4. Wait to be redirected into the application.

C.2 Create Assembly

Starting point

The user has logged into the application and is located on the *systems* page.

Steps

1. Locate the Test system in the systems table and click on the "assemblies" button.
2. Wait to be redirected to the assemblies page.
3. Locate the "Add Assembly" button at the top right corner and click on it. A popup form should appear.
4. Fill out 'Test assembly' in the assembly name field.
5. Fill out 'TSA' in the abbreviation field.
6. Click the submit button in the bottom right of the dialog window and wait for the dialog window to close.

C.3 Attach File to Assembly

Starting point

The user is logged into the application and is located on the *assemblies* page of the Test System. The assemblies table contains one entry with the name 'Test Assembly'. The user was provided with a test image.

Steps

1. Locate the Test assembly in the assemblies table and click on the document icon button in the corresponding row. A dialog window called File Manager should open.
2. Click on the input file form field and wait for the system file manager to open.
3. Select the test image from the disk.
4. Click on the "Add File" button and wait for the file to be uploaded.
5. Confirm that the file was uploaded and the entry is showcased in the dialog.
6. Close the File Manager dialog window.

C.4 Create Part

Starting point

The user is logged into the application and is located on the *parts* page of the Test Assembly.

Steps

1. Click on the "Add Part" button located in the top right corner. Wait for the dialog window to open.
2. Fill out 'Part name1' in the name field.
3. Leave the Comment field blank.
4. Check the part as made.
5. Click on the "Submit" button in the bottom left corner and wait for the dialog window to close.
6. Check that the data were correctly updated.

C.5 Edit Part

Starting point

The user is logged into the application and is located on the *parts* page of the Test Assembly. In the parts table is one entry named 'Part name1'.

Steps

1. Locate the part with the name Part name1 in the parts table and click on the ”Edit” button in the corresponding row. Wait for the dialog window to be opened.
2. Change the part name to ’Part name’ in the name field.
3. Check the part as bought.
4. Click on the ”Submit” button in the bottom left corner and wait for the dialog window to close.
5. Check that the data were correctly updated.

C.6 Create Part Detail

Starting point

The user is logged into the application and is located on the *parts* page of the Test Assembly. In the parts table is one entry named ’Part name1’.

Steps

1. Locate the part with the name Part name1 in the parts table and click on the ”Details” button in the corresponding row. Wait for the dialog window to be opened.
2. Locate the ”Add new Detail” button and wait for a dialog window to be opened.
3. Choose ’material’ in the type field dropdown.
4. Fill ’steel’ in the name field.
5. Fill 3 in the quantities field.
6. Fill 15.26 in the cost field.
7. Leave the comment and cost comment fields empty.
8. Click on the ”Submit” button in the bottom left corner and wait for the dialog window to close.
9. Check that the data were correctly updated.

C.7 Generate BOM

Prerequisite

The user is logged into the application and is located on the *systems* page.

Steps

1. Click on the ”Generate BOM” button located in the top right corner of the screen.
2. Wait for the document to be generated and downloaded.
3. Open the downloaded document and verify that all of the previously input information is present.

C.8 Generate Support File

Prerequisite

The user is logged into the application and is located on the *systems* page.

Steps

1. Click on the ”Generate Support File” button located in the top right corner of the screen.
2. Wait for the document to be generated and downloaded.
3. Open the downloaded document and verify that the document contains the inputted test image with the correct custom ID and system's abbreviation.

C.9 Delete Assembly

Prerequisite

The user is logged into the application and is located on the *systems* page.

Steps

1. Locate the Test system in the systems table and click on the ”Assemblies” button in the corresponding row. Wait to be redirected to the assemblies page.
2. Locate the Test Assembly in the assemblies table and click on the ”delete” button in the corresponding row. Wait for the confirmation dialog to be opened.
3. Click on the ”delete” button and wait for the confirmation dialog to be closed.
4. Confirm that the assemblies table is now empty.

C.10 Logout

Prerequisite

The user is logged into the application.

Steps

1. Click the ”logout” button at the bottom of the navigation bar on the left-hand side of the application.
2. Wait to be redirected to the login page.

Appendix D

Contents of attachments

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
  - `screenshots` ..... illustraisions of the developed application
- `src` ..... the directory of source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format