Development of a platform for building design, validation, and optimization of modules transportation

Enzo Dornelles Italiano - 52549

Project report presented to the School of Technology and Management of Bragança to obtain the master's degree in Informatics within the scope of the double degree program with the Federal University of Technology - Paraná.

Supervisors:

Paulo Alexandre Vara Alves

José Eduardo Fernandes

Juliano Henrique Foleis

Bragança

October 2023

Development of a platform for building design, validation, and optimization of modules transportation

Enzo Dornelles Italiano - 52549

Project report presented to the School of Technology and Management of Bragança to obtain the master's degree in Informatics within the scope of the double degree program with the Federal University of Technology - Paraná.

Supervisors:

Paulo Alexandre Vara Alves

José Eduardo Fernandes

Juliano Henrique Foleis

Bragança

October 2023

Acknowledgment

I would first like to express my gratitude to the Federal University of Technology - Paraná (UTFPR) and Institute Polytechnic of Bragança (IPB) for providing me the chance to take part in this program.

In second place, the patience and availability of the supervisors of the project, Prof. Paulo Alves, Prof. José Eduardo Fernandes, and Prof. Juliano Foleiss, had during this project.

Finally, I'd like to thank my family and friends for their advice and support throughout difficult decisions and struggles.

Abstract

In a world where each day is becoming more intertwined with technology, numerous companies strive to discover solutions for new project developments or enhance existing complex systems. Their aim is to reduce material wastage, energy consumption, labor dependency, project timelines, and overall costs.

A recent trend in civil construction is to use pre-fabricated components such as walls, beams, columns, and other components. These components can be implemented in many different types of buildings and along with technology the system can save information about them, preventing risks during the construction, reducing costs for the companies, and improving the efficiency of the company as also being the producers of the components.

The development of this API was carried out using the Python programming language, which facilitated rapid development. It leveraged the FastAPI framework, offering the flexibility to create diverse methods for validating structural and architectural regulations for buildings. The API also provides an estimate of the required pre-fabricated components for construction.

Additionally, it incorporates optimization for cargo loading, a pivotal aspect of civil engineering. A genetic algorithm is employed to identify the optimal solution for fitting components within a container. This not only enhances delivery times but also reduces costs by avoiding unnecessary containers.

As a comprehensive system designed for the market, this API includes endpoints for essential web app functionalities. It aims to prevent user errors and has been rigorously

tested and approved by architects specialized in buildings featuring pre-fabricated components. Any identified errors have already been rectified.

Keywords: Modular Construction, Cargo Loading Optimization, Civil Engineering, Hybrid Genetic Algorithm.

Resumo

Num mundo onde a tecnologia está cada vez mais interligada com o nosso dia a dia, muitas empresas estão em busca de soluções para novos projetos ou para aprimorar sistemas complexos já existentes. O objetivo é reduzir o desperdício de materiais, o consumo de energia, a dependência de mão de obra, os prazos de projeto e os custos gerais.

Uma tendência recente na construção civil é o uso de componentes pré-fabricados, como paredes, vigas, colunas e outros elementos. Esses componentes podem ser aplicados em diversos tipos de edifícios e, juntamente com a tecnologia, o sistema pode armazenar informações sobre eles, prevenindo riscos durante a construção, reduzindo custos para as empresas e melhorando sua eficiência, bem como a eficiência dos produtores de componentes.

O desenvolvimento desta API foi realizado usando a linguagem de programação Python, o que possibilitou um desenvolvimento rápido. Ela aproveitou o framework FastAPI, oferecendo a flexibilidade para criar diversos métodos de validação de regulamentos estruturais e arquitetônicos para edifícios. A API também fornece uma estimativa dos componentes pré-fabricados necessários para a construção.

Além disso, ela incorpora a otimização para o carregamento de carga, um aspecto fundamental na engenharia civil. Um algoritmo genético é utilizado para identificar a solução ideal para acomodar os componentes dentro de um contêiner. Isso não apenas melhora os prazos de entrega, mas também reduz os custos, evitando contêineres desnecessários.

Como um sistema abrangente projetado para o mercado, esta API inclui endpoints para funcionalidades essenciais de aplicativos da web. Seu objetivo é evitar erros do

usuário e ela foi rigorosamente testada e aprovada por arquitetos especializados em edifícios com componentes pré-fabricados. Quaisquer erros identificados já foram corrigidos.

Palavras-chave: Sistemas de Recomendação, Construção Modular, Otimização de Carregamento de Carga, Engenharia Civil, Algoritmo Genético Híbrido.

Contents

List of Tables

List of Figures

Acronyms

AEC Architecture, Engineering and Construction.

API Application Programming Interface.

BIM Building Information Modeling.

CeDRI Research Centre in Digitalization and Intelligent Robotics.

CLP Container Loading Problem.

ESTiG Escola Superior de Tecnologia e Gestão.

GA Genetic Algorithm.

IPB Institute Polytechnic of Bragança.

SOA Services Oriented Architecture.

UTFPR Federal University of Technology - Paraná.

Chapter 1

Introduction

This chapter gives an overview of the background and motivations of this project. Then, the objectives are presented, as well as the document structure.

1.1 Overview

The current work aims to develop an Application Programming Interface (API) related to modular construction with prefabricated components. It is part of the MICADO project of the Research Centre in Digitalization and Intelligent Robotics (CeDRI) (CeDRI & IPB, 2021). The MICADO Project consists of developing a modular composite solution for constructing rationalized reinforced concrete structures, highly prefabricated and with reduced waste, intending to increase energy performance and reduce the environmental impacts of the life cycle (AMA, 2023). To fulfill MICADO's requirements and be effective, the backend encompasses the system's constraints, structural and architectural rules, and the logistics module for component delivery.

It is crucial for the project that the rules and all validations provide the values accurately because a mistake could result in the collapse of an entire structure, which could result in accidents. So, all the possibilities must be evaluated to prevent eventual errors.

This work documents the steps of the development followed, explaining the validation functions, the recommendation system for delivering components, and extra features

included in the final release.

1.2 Objectives

The objective of the current work is to establish an API that enables the design and validation of secure and cost-effective modular constructions tailored to the specific requirements of each building. The objectives shall be achieved were:

- Develop a strategy to validate architectural principles
- Develop a strategy to validate structural principles
- Optimize the distribution of the components for transportation;

The strategy for validating architectural and structural principles was idealized to prevent errors, be easy to maintain, and be easily improved in the future if necessary for any upgrade in the platform.

Since it is an entire web application and numerous actors are participating in the project, functionalities related to modular construction are only some of the ones required. Access control for each actor is also crucial to secure routes that require a specific user level.

The users' level is highly sensitive information, and any process that requires it must be thoroughly tested before deployment. This type of attack is common in web services and can disrupt the system, impacting all users.

One of the challenges here is to guarantee that all the services will work appropriately according to their requirements and permitted users. An error in the structural rules may cause severe injuries to the house owner and the building company, making any mistakes unacceptable.

The main challenge is the recommendation system for the distribution of the components, which must incorporate various shapes and be optimized for various modes of transport while minimizing unneeded expenses.

1.3 Document Structure

This document is divided into six chapters. The current chapter presents an introduction with an overview and the goals to achieve. The rest of the document is organized as follows:

- Chapter 2 presents the literature review;
- Chapter 3 describes the requirements in more detail;
- Chapter 4 presents the implementation, highlighting difficulties and technical solutions for the most important problems;
- Chapter 5 brings the tests used to validate the system if it solves the requirements presented in Chapter 3;
- Chapter 6 concludes the discussion and brings suggestions for future work.

Chapter 2

Context

This chapter presents concepts such as the Building Information Modeling (BIM) technology with prefabricated buildings, and container loading problems. It also presents what the market already has related to building design applications.

2.1 BIM technology

Prefabricated buildings have been widely studied to change how buildings are made, reducing material waste, energy consumption, labor required, project duration, and costs. All these advantages can be achieved using components prefabricated in factories, such as walls, beams, columns, and other components (Haas et al., 2002).

In advance, to design this kind of building, the BIM is the primary methodology used to preserve information about every single component, from the project conception until its demolition, used at various phases of the construction lifecycle. Its integration into construction regulations has already been implemented in various European countries (BIMTech, 2016).

The mechanical, electrical, and plumbing components of a building are included in BIM technology. Na Lu and Thomas Korman (Lu & Korman, 2010) discuss the advantages of using BIM for construction in order to decrease process fragmentation between design and construction firms, streamline technology between engineers and construction contractors,

and, finally, provide a model for use by specialty contractors planning prefabricated.

BIM's benefits are better applied to new projects because they can be implemented more effectively as all the component information is readily available. The prevalence of outdated, insufficient, or fragmented information in existing buildings can lead to inefficient building management, increasing costs and adding time to many processes. BIM is anticipated to have limitations due to missing information throughout the building process (Volk et al., 2014).

Even with many advantages, some factors still influence BIM being unused for all types of industries, as related in 2017 (Sun et al., 2017). Due to the advantages, many businesses now invest in BIM technology to lessen drawbacks like the expense of specialized software, as this work is an example.

BIM is not just a technology change but also a process change. Representing a building with intelligent objects carrying its information and its behavior with each other may change the critical process of the building conception: how early-stage concepts and area plans are created using the client's programmatic requirements; how design options are evaluated for factors like energy use, constructability, expense, and way-finding; how a multidisciplinary team of designers works together to create a design; how the building is built, including how different components are made by subcontractors; how the building facility is operated and maintained after construction. New capabilities are brought up along with improving these processes, catalyzing significant processes and contractual changes in the Architecture, Engineering and Construction (AEC) industry (ISO, 2016).

2.2 Building Design Applications

Websites and applications for building design must also be mentioned as an inspiration of what kind of software already exists with similar purposes.

A platform called Homestyler (Easyhome New Retail Group & Alibaba Group, 2009) was created by Alibaba and Easyhome New Retail Group. This platform enables the design of a house, including the interior and exterior spaces, using a wide variety of

elements and textures with a contemporary style and numerous lighting visualizations based on the environment of choice. From each section, rendered images can also be downloaded. There are subscriptions available on the website for unlimited renders and the release of more complicated models; the platform is not licensed as open source.

A BIM software called Revit (Autodesk, 2002) was created by the American multinational software giant Autodesk. As the proprietor of AutoCAD, the world's most popular 2D CAD program, Autodesk offers Revit as a supplemental product targeted at the building construction sector. Revit is an effective tool for organizing, designing, constructing, and managing construction projects because it was created specifically for creating parametric 3D virtual models of building components (Farias, 2019). It integrates seamlessly with other Autodesk products like Navisworks and BIM 360, making it an excellent option for multidisciplinary teams working together on challenging construction projects. This program has become crucial to the modern construction sector.

The SketchUp (Trimble Inc., 2023) software can also be considered BIM due to its interoperability, Industry Foundation Classes (IFC) support, dynamism, structure, data extraction, and cost estimation (Barsa, 2021). It was launched in 2000 by At Last Software. Six years later, it was sold to Google, and in 2012, it was sold to Trimble with some new versions and improvements. The software has interactions and complements with other programs such as AutoCAD, Revit, Vray, Lumion, and other options described in articles on the SketchUp blog, which can share files or give new features like pictures of the project (Razor, 2019).

React Planner ([CVDLAB], 2019) is a building-specific drawer React component created by RomaTre University's Computational Visual Design Lab (CVDLAB). The platform enables a 3D representation of the drawn plan and home design with a predetermined catalog of ready-to-use objects. A developer can expand the catalog with new products. Currently open source and covered by the MIT License, it was used as a base version for the front end of the API described in this work.

2.3 Transportation in Civil Construction

Transportation of construction materials represents around 20% of the total cost in typical industrial construction projects. Even though companies usually give little attention to the transportation stage, causing delays in the delivery (F.F. et al., 2014).

The importance of this module is that the transport of any cargo represents many risks for companies around the globe. They can be the fall of the cargo itself or the transport vehicle, something hit the cargo, the vehicle, people, or their property, causing inconveniences for the transporter and the other people using the highway. Climate conditions, geometry, and weight of the loads may also cause accidents (Navarro, 2012), these many variables make companies invest in research to find alternatives and reach the best option suffering less impact for all the people involved in the process.

Portugal did not prioritize having many highways throughout its territory until the 1980s; they only started to focus on improving the infrastructure in need, after they joined the European Union in 1986 (Mónica Silvares, 2021). In this context, the studies in this area are very new, impacting civil construction.

The role of infrastructure and its contribution to economic development was highlighted primarily by all Portuguese and Galician experts in their responses to issues directly related to this matter. The investments in this area not only have the objective of facilitating and expediting the movement of people and goods, but, as confirmed by experts from the Galician Administration, it has immediate impacts on regional income, increased employment, particularly in the construction sector, reduced costs, and increased efficiency of companies and population mobility (Jorge Evaristo Cuntín Rey et al., 1999). +

Chapter 3

Modeling

This project has a specific document for the requirements of the system describing each feature, actor, and architecture of the system. Its guidelines are described in this chapter, which were followed throughout the development.

3.1 System Architecture

The project includes not only the backend described in this paper but also a front-end interface to interact with.

Figure 3.2 presents all the modules the project was planned to contain. For the front-end, a 2D and 3D visualization, budgets, orders, and logistics. For the backend, system limits, architectural rules, and logistics. It also suggests a No-SQL database.

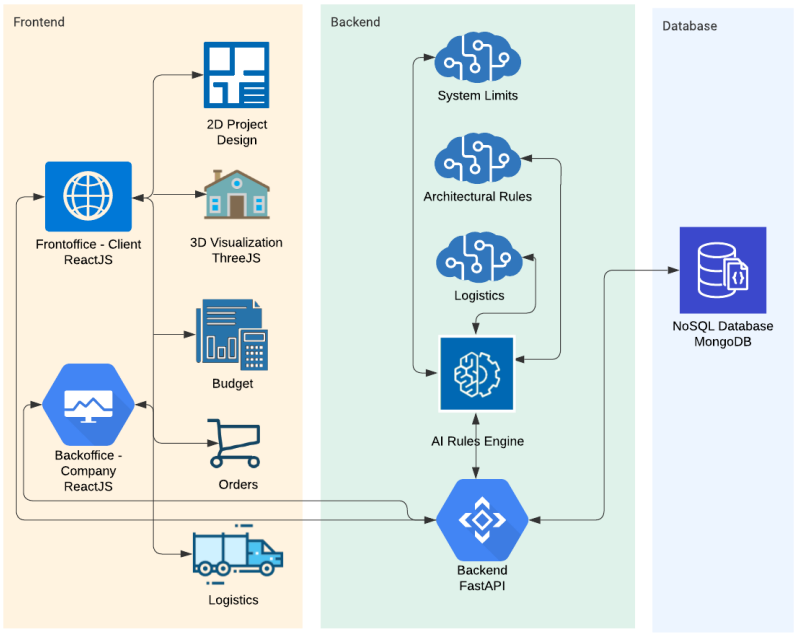

Figure 3.1: General Architecture of the Platform

The 2D view is used for creating and editing the building with the available resources, such as walls, windows, doors, and furniture. It is the interface where the user verifies the errors with its construction and fixes them. The 3D visualization gives an idea of the construction after it is finished.

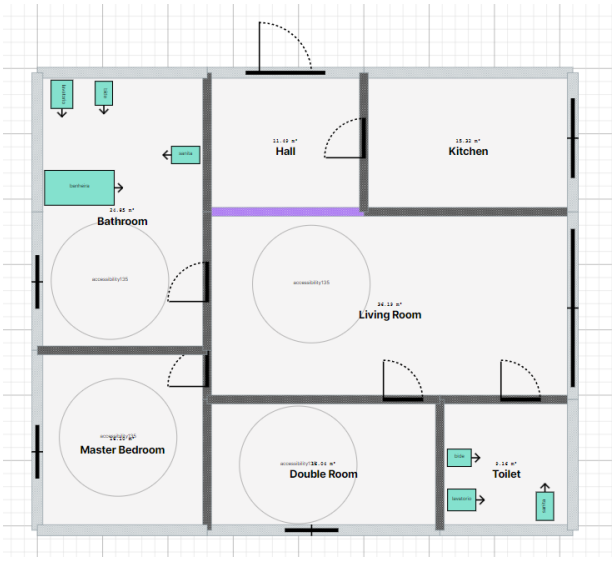

Figure 3.2: 2D Visualization

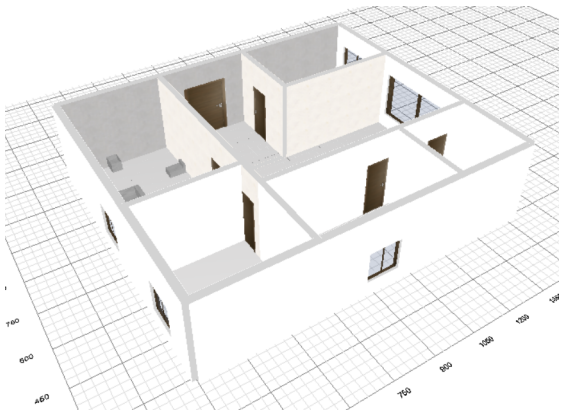

Figure 3.3: 3D Visualization

The budget, orders, and logistics are connected by the process to which they belong. The budget page of a building shows the necessary pre-made modules, their components, and their prices. After accepting the budget, an order is created, and the system can optimize the packages for transportation logistics.

PEÇA	COMPONENTE	UNIDADES	VAL. UNIT	TOTAL
CJ.CAN	Ar. Can	12	On request	On request
CJ.CAN	Pn. Can	12	On request	On request
CJ.CAN	GA CAN	12	On request	On request
CJ.ZCR 1200	Ar. 1200	18	On request	On request
CJ.ZCR 1200	Pn. 1200	18	On request	On request
CJ.ZCR 1200	GA 1200	18	On request	On request
CJ.ZCR 600	Ar. 600	3	On request	On request
CJ.ZCR 600	Pn. 600	3	On request	On request
CJ.ZCR 600	GA 600	3	On request	On request

Figure 3.4: Components list for the budget of the building

As the main objectives of this work, the backend components received more attention in detail. The system limits component is related to structural rules that the building

must follow. The objective of this component was to validate the rules and find the best solution to overcome impossibilities. In this case, the obstacle is to verify if the design of the building, done by the customer, can support the materials used in the construction.

For the architectural rules component, most of the rules received by the product owner were related to the Decree-Law n.º 38382 of 7^{th} August 1951 and its alterations as the project is based in Portugal needs. This decree describes fixed rules that must be followed by every building inside Portugal with specifications for each area, doors and windows dimensions, number of furniture, and other rules.

The logistics component involves budget calculation with the components in use in the building to optimize the organization of the packages for transportation and control the status of the orders in the system. The budget's calculation goal is to give the user an ending price so they can choose whether or not to move forward with the construction.

The order's status is not precise at the moment, as it depends on the company responsible for the transport. However, it is already possible to set a date manually for the user to have access to this information. The optimization of the packages for transportation is described in another section in this chapter.

The idea for these backend components was to create something easy to maintain and to incorporate new rules if necessary in the future. Both are hard coded but with variables that can be easily modified for different values. In addition, this might be fine for architectural rules as they must follow the country's law and the specifications take some time to change.

Regarding code structure, the API is currently monolithic but split into categories that can be converted to distinct services if a Services Oriented Architecture (SOA) is necessary. The structure in use allowed faster development at the beginning due to its simplicity, and it did not change later.

3.2 Actors

Describing actors is an essential stage of any software development, helping to identify the different roles that users and systems play in the software's ecosystem. It aids in comprehending the roles of the various actors and how the system will interact with them. It also helps recognize potential risks and difficulties that can appear throughout the development process and how they can be avoided. Finally, it enables the development of a user-centric system by enabling software to be designed following user needs and expectations.

In Figure 3.5, it is possible to identify each kind of user of the system, including visitor, who does not need to authenticate on the website but also has limited permissions, and the other actors who need authentication are the client, project technician, sales technician and the administrator of the system.

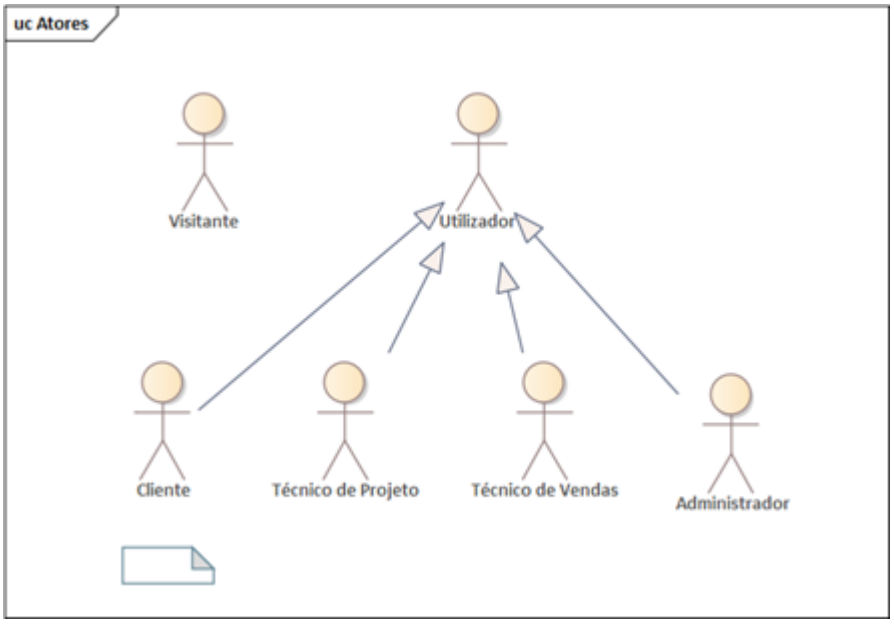

Figure 3.5: Actors of the system

- The visitor can access the company's public information and information about its projects, services, and modular buildings. Additionally, they can sign up for the platform.

- The client role has access to the main services of the platform, being able to follow the whole process of its building.
- The project technician is the one who is going to validate the technical requirements of the created projects according to the structural and architectural rules.
- The sales technician is responsible for managing the orders.
- The administrator is the system owner who has access to manage construction modules and associated rules and to view all information related to projects, budgets, and orders.

3.3 User stories

The user stories were established at the start of the project and were in charge of outlining the requirements and functionalities. They serve as an illustration of what the software can accomplish once the process is complete. The end user's expectations can be better understood by approaching this from their point of view as the user. They served as a method to prioritize tasks so the team could concentrate on what was more crucial to the project while also assisting with communication and project management.

Each actor has its specifications described using user stories. Here are listed the relevant stories to the backend development.

Table 3.1: Backend User Stories for Visitants

User Story	User Level
As a Visitor, I want to register on the platform	Visitor
As a Visitor, I want to log in to have a profile on the platform	Visitor

The user stories for visitors are only related to registering and logging in. It is an essential feature for access control and identifying the other system roles.

Table 3.2 represents the user stories for the customer. These user stories are essential functionalities for testing the main objectives of this project, as the stories include

validating the building created by the user against the system limits and architectural rules.

Table 3.2: Backend User Stories for Customer Role

User Story	User Level
As a Client, I want to create a new housing project	Client
As a Client, I want to save the housing project	Client
As a Client, I want to open a previously saved housing project	Client
As a Client, I want to automatically validate the model against the system limits	Client
As a Client, I want to automatically validate the model against architectural rules	Client
As a Client, I want to obtain the budget for the housing, with a breakdown of the necessary modules	Client
As a Client, I want to place an order for the necessary building modules for the housing, with an estimated delivery date	Client
As a Client, I want to know the status of the order, specifically whether it has been validated, the modules have been ordered from the supplier, in transit, shipped, and received	Client

The following Table 3.3 presents the user stories for the project technician role. These stories are essential for the continuation of the validation project of the building designed by the user and also for checking information necessary for the users to decide if they are going to make changes in the design or give up the construction due to its cost.

Table 3.3: Backend User Stories for Project Technician Role

User Story	User Level
As a Project Technician, I want to view the projects of clients awaiting validation	Project Technician

As a Project Technician, I want to open a project to analyze the system and architectural rules	Project Technician
As a Project Technician, I want to validate a project in terms of how it complies with system and architectural rules	Project Technician
As a Project Technician, I want to change a project so that it can comply with system and architectural rules	Project Technician
As a Project Technician, I want to obtain a list of modules needed for the project	Project Technician
As a Project Technician, I want to consult the housing budget	Project Technician
As a Project Technician, I want to know the status of the order, namely if it has already been validated, the modules have been ordered from the supplier, in shipment, sent and received	Project Technician
As a Project Technician, I intend to access the budget for the housing.	Project Technician

Table 3.4 has an essential story for achieving one of the objectives: obtaining the optimized distribution of the packages for transportation. It can also depict the sales technician's roles and responsibilities in delivering the modules for building construction.

Table 3.4: Backend User Stories for Sales Technician Role

User Story	User Level
As a Sales Technician, I want to view the projects.	Sales Technician
As a Sales Technician, I intend to open a project to view the necessary modules.	Sales Technician
As a Sales Technician, I aim to obtain a list of the required modules.	Sales Technician
As a Sales Technician, I want to check the status of the order.	Sales Technician
As a Sales Technician, I aim to change the status of the order.	Sales Technician

As a Sales Technician, I want to obtain the optimized distribution of the load (modules) for transportation.	Sales Technician

The Table 3.5, representing the stories for admin users, has some other tables and includes management features to control the system.

Table 3.5: Backend User Stories for Administrator Role

User Story	User Level
As an Administrator, I want to view the projects of the clients.	Administrator
As an Administrator, I aim to manage a project.	Administrator
As an Administrator, I want to obtain a list of the modules required for the project.	Administrator
As an Administrator, I aim to manage the construction modules.	Administrator
As an Administrator, I want to access the budget for the housing.	Administrator
As an Administrator, I intend to manage the budget for the housing.	Administrator
As an Administrator, I aim to know the status of the order, specifically whether it has been validated, the modules have been ordered from the supplier, in transit, shipped, and received.	Administrator
As an Administrator, I intend to manage the order.	Administrator
As an Administrator, I want to obtain the optimized distribution of the load (modules) for transportation.	Administrator

3.4 Database Collections

Even using a non-relational database, it was structured with everything split to comprehend its collections better.

3.4.1 Contents

The *Contents* collection stores the most important information about the building, everything necessary for the front end to generate the image for the user.

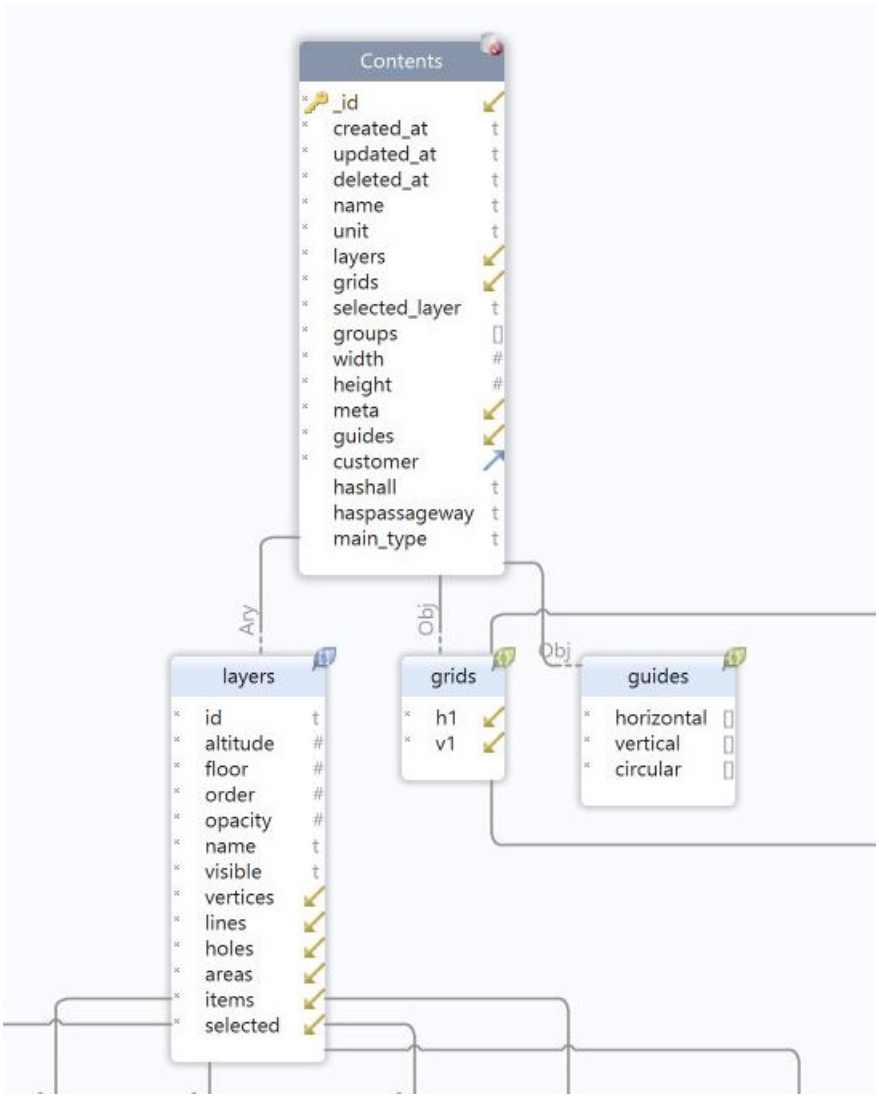

Figure 3.6: Contents Collection Diagram with major properties

Some of the properties of the document are easy to understand simply with the name, such as *name*, *unit*, *customer*, *hashall*, *haspassageway* and *main_type*.

The property *layers* saves each floor of the building with its information of altitude, number of the floor, order, opacity, name, and if it is visible. The properties of vertices, lines, holes, items, and areas have information about each item to be drawn to the user. At last, the *selected* property saves if any of the items were selected when the project is closed.

The *grid* property has the values of the grid, which the user can define. The same for the *guides* property.

3.4.2 Graphs

The graphs are the most complex collection of the system. It has pre-processed information of the *Content* collection of the project to which it belongs. It simplifies or calculates information about vertices, lines, areas, holes, and items, making it easier to validate the construction. It also has information on passageways, halls, and the possibility of construction.

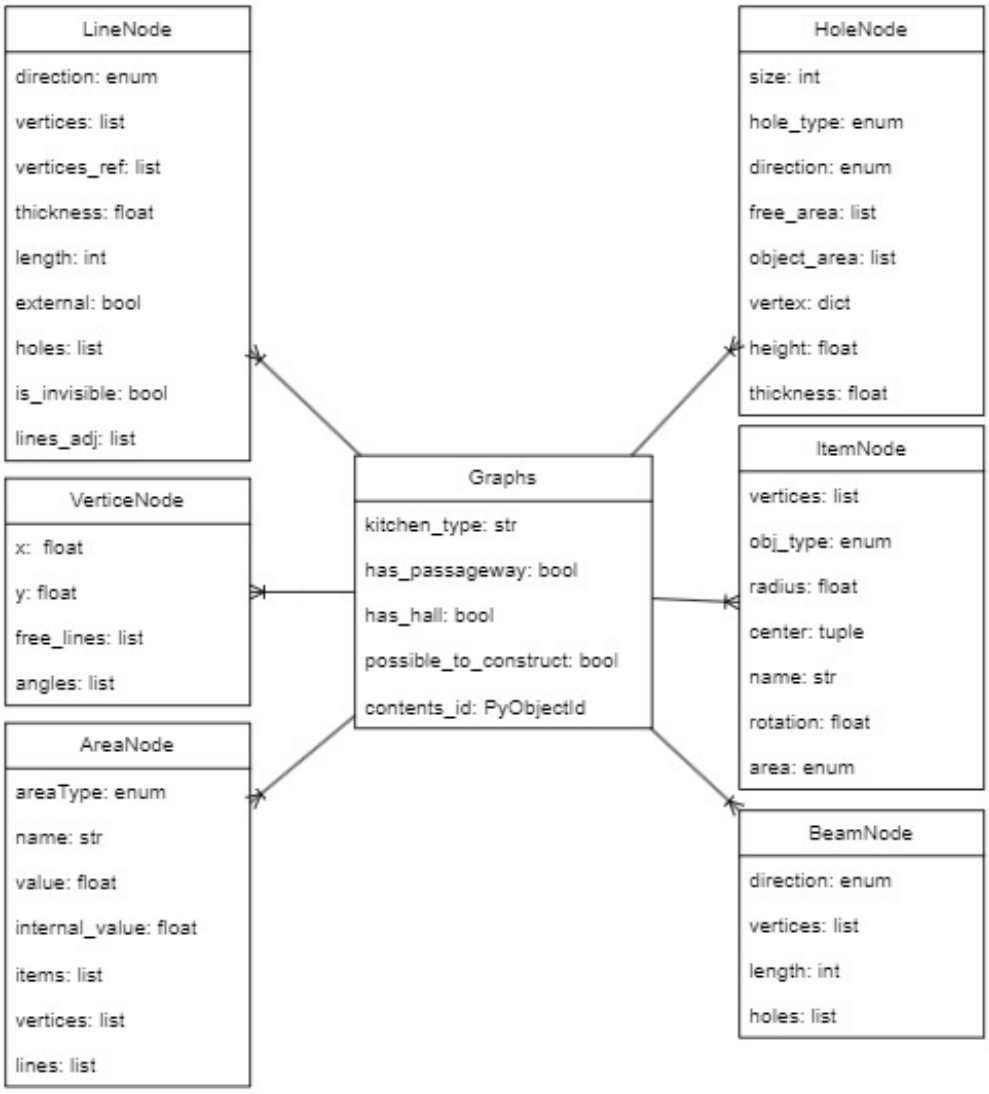

Figure 3.7: Graph Diagram

All the vertices of the building are stored with its coordinates, directions it does not have lines, and the angles it has with its adjacent.

```python
class VerticeNode(BaseModel, extra=Extra.forbid):
    x: float
    y: float
    free_lines: List[int] = []
    angles: List[Tuple[str, str, float]] | None
```

Figure 3.8: Vertice Node Class

The lines of the building are stored with information about its orientation, vertices, thickness, and length; if it represents an external wall; if it is an invisible wall, the adjacent lines, and the holes it has in it.

```python
class LineNode(BaseModel, extra=Extra.forbid):
    direction: DirectionType | None
    vertices: List[Tuple]
    vertices_ref: List[str] = []
    thickness: float = None
    length: int
    external: bool
    holes: List[str] = []
    is_invisible: bool = None
    lines_adj: List[str] = []
```

Figure 3.9: Line Node Class

For the areas, the properties stored are the area type, area's value, and the area's internal value, which considers the thickness of the wall, the items the area has, its vertices, and its lines.

```python
class AreaNode(BaseModel, extra=Extra.forbid):
    areaType: AreaType
    name: str
    value: float
    internal_value: float | None
    items: List[str] | None
    vertices: List[str]
    lines: List[str] | None
```

Figure 3.10: Area Node Class

The items have properties that vary depending on the type of the object. For accessibility items, it is necessary to have the radius and the center coordinates. For regular objects, the vertices of them are stored, and also the rotation. All of the objects have a name and its type.

```python
class ItemNode(BaseModel, extra=Extra.forbid):
    vertices: List[Tuple] | None
    obj_type: ObjType | None
    radius: float | None
    center: Tuple | None
    name: str
    rotation: float = None
    area: AreaType = None
```

Figure 3.11: Item Node Class

For the holes, it is stored its size, type, orientation, height, thickness, the coordinate of its beginning, the free area, which is more important for doors, and the area of the hole.

```python
class HoleVertex(BaseModel, extra=Extra.forbid):
    x: float
    y: float


class HoleNode(BaseModel, extra=Extra.forbid):
    size: int
    hole_type: HoleType | None
    direction: DirectionType
    free_area: List[Tuple]
    object_area: List[Tuple]
    vertex: HoleVertex
    height: float | None
    thickness: float | None
```

Figure 3.12: Holes Node Class

The beams only have to store their orientation, starting and ending coordinate points, length, and the holes they can have.

How all this information is inferred from the *Content* collection is explained in section 4.

3.4.3 Templates

The templates have the same information as the *Content* collection, except for the customer ID, which is not necessary as the administrator of the system creates them.

The objective of having templates is to provide ready-made projects already validated for the user to adjust as desired and to speed up the design process.

3.4.4 Customers

Basic customer information is used to identify users of the platform, including name, fiscal identification number, role, and the login requirements of username, email address, and password.

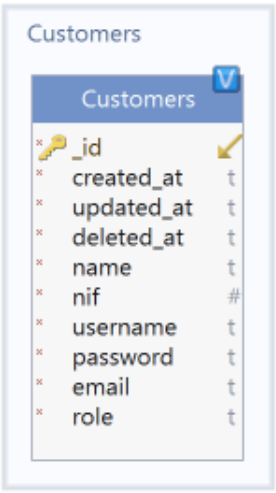

Figure 3.13: Customers Collection Diagram

The customers are the actors of the system, meaning a very important collection for the correct work of the system. It is a usual collection to have in every system nowadays to make access control to certain types of content, commonly used in web apps.

3.4.5 Modules

These are the modules used in construction. However, the collection is solely used to determine how much the structure would cost because it contains information on how

much each module is worth. As a result, it includes the module's name, value, name of each component, number of units in the module, reference number, and description.

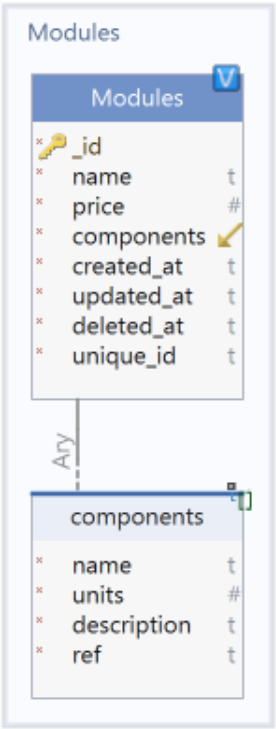

Figure 3.14: Modules Collection Diagram

The definitions and properties that actually affect the design on the frontend are stored in the Catalog collection. The Modules collection has more impact on the budget and logistics components of the system.

3.4.6 Projects

This collection is in charge of keeping track of the project's status and a delivery forecast. It contains the identifiers for the content, the project reviewer, and the owner.

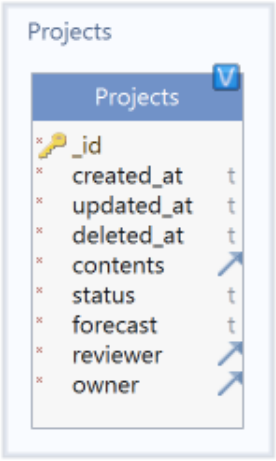

Figure 3.15: Projects Collection Diagram

Its attributes are very similar to the Requests Collection, but this one is related to the approval of the project technician for the building to be constructed and sent to the sales technician.

3.4.7 Requests

An essential collection to give the user a way to verify the status of the process of each of its building designs.

A new request document is created when the user makes a new order. As a result, this collection contains the order status, the customer identifier, and the content.

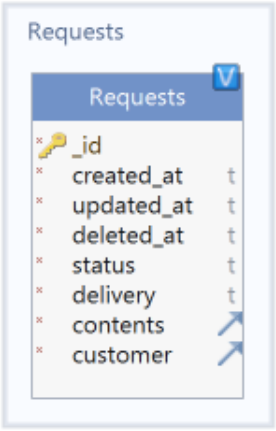

Figure 3.16: Requests Collection Diagram

3.4.8 Catalog

The documents of this collection act as a filter for the catalog of the items that the user can use on its building, and the administrator is responsible for turning items on and off.

The properties of this collection are related to information necessary for the front end to render each item, being the name, prototype, the name of the MTL file, OBJ file, and the texture, if it is enabled, and the category it is going to belong. Other information includes width, height, altitude, thickness, title, tags, and description.

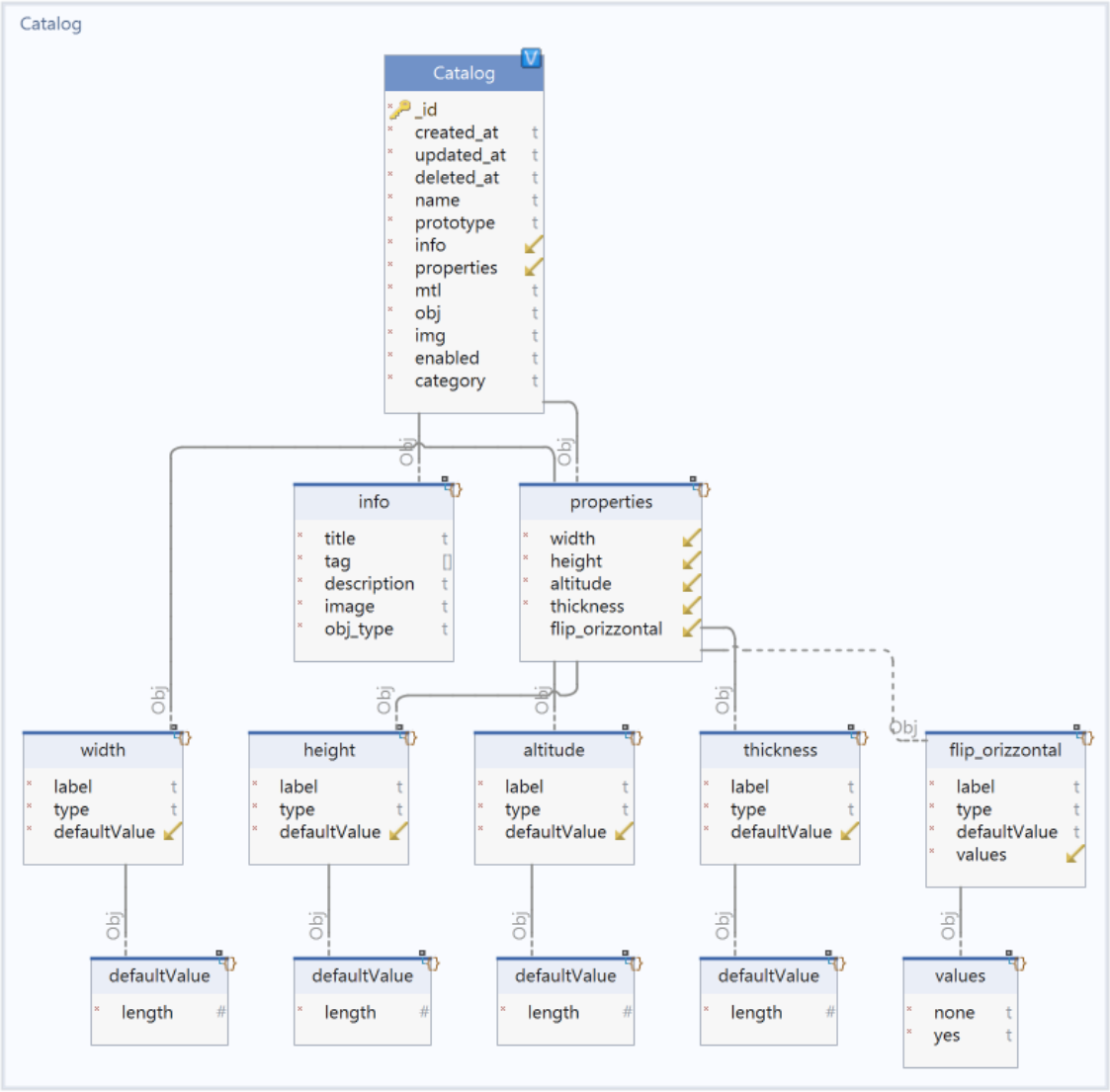

Figure 3.17: Catalog Collection Diagram

3.5 Container Loading Problem

For the logistics part of the system, the last objective to describe is the optimization of the packages for transportation, which consists of fitting the packages inside a container or a truck.

For this kind of problem, there are three classifications we could use to solve the organization of the pieces (Martello et al., 1998).

- Knapsack loading. Each box in the knapsack loading of a container has a corresponding profit, and the challenge is to select a subset of the boxes that will fit into a single container (bin) to load the most profitable boxes. The goal is to increase the chosen items' total profit while ensuring their combined volume does not exceed the container's holding capacity. Since the objective is to pack the items as tightly as possible to minimize the amount of empty space in the container, if the profit of a box is set to its volume, this problem corresponds to the minimization of wasted space;

- Container loading. The container loading problem is a particular kind of packing problem where every box must be crammed into a single, infinitely long container (bin). Finding a workable solution that minimizes the length of time the bin is filled while ensuring that the combined volume of the boxes does not go over the bin's capacity. This issue is fundamental in logistics and transportation since goods must be transported effectively and affordably. Finding the best solution for larger instances of the container loading problem is computationally difficult because it is an NP-hard problem. However, several heuristic and metaheuristic algorithms can deliver accurate approximations in a respectable amount of time;

- Bin packing. In order to solve the 3D-BPP, all of the boxes must be packed into the bins, but unlike the container loading problem, all of the bins have finite dimensions, so the goal is to find a solution that uses the fewest number of bins.

In the current work, to solve the Container Loading Problem (CLP), a hybrid Genetic Algorithm (GA) with the bin packing approach, as the container's or container's dimensions are already defined. A diploid chromosome is used as a representation for individuals where they represent the order and the rotation of the boxes in the container. A construction heuristic called Simple Deepest Bottom Left with Fill (S-DBLF) transforms an individual into a 3D solution. S-DBLF moves each box in a set of boxes to the

deepest available position, then as much to the bottom as possible, and eventually as far to the left as possible. The algorithm includes two checks: the box should not exceed the dimensions of the position, and it should not overlap with any of the previously placed boxes. Finally, the NSGA-II algorithm ranks the solutions and selects the best ones that converge near the true Pareto-optimal set (Ramesh & Umashankar, 2021).

3.6 System Rules

The document with the rules for the system was provided by the product owner and includes architectural and structural rules to follow. It is divided into general rules and types of spaces with architectural rules associated. Here in this paper, is divided into 3 parts: structural rules and restrictions, architectural rules and restrictions, and types of spaces with architectural rules associated for better identification of the rules.

3.6.1 Structural Rules and Restrictions

The structural rules are the most important of the system, as they may cause the collapse of the building as they are related to the material used in the construction.

Clear Span of Slabs

In at least one of the directions, the corner/edge pieces must be at a distance equal to or less than 7m. This criterion is imposed because the slabs cannot span a distance greater than 7m.

The slab will discharge in the direction where there is a shorter distance between walls. In the event that in both directions the distance between walls is less than 7m and according to the following criteria presented, the slab can be placed to discharge in the direction with the greater separation.

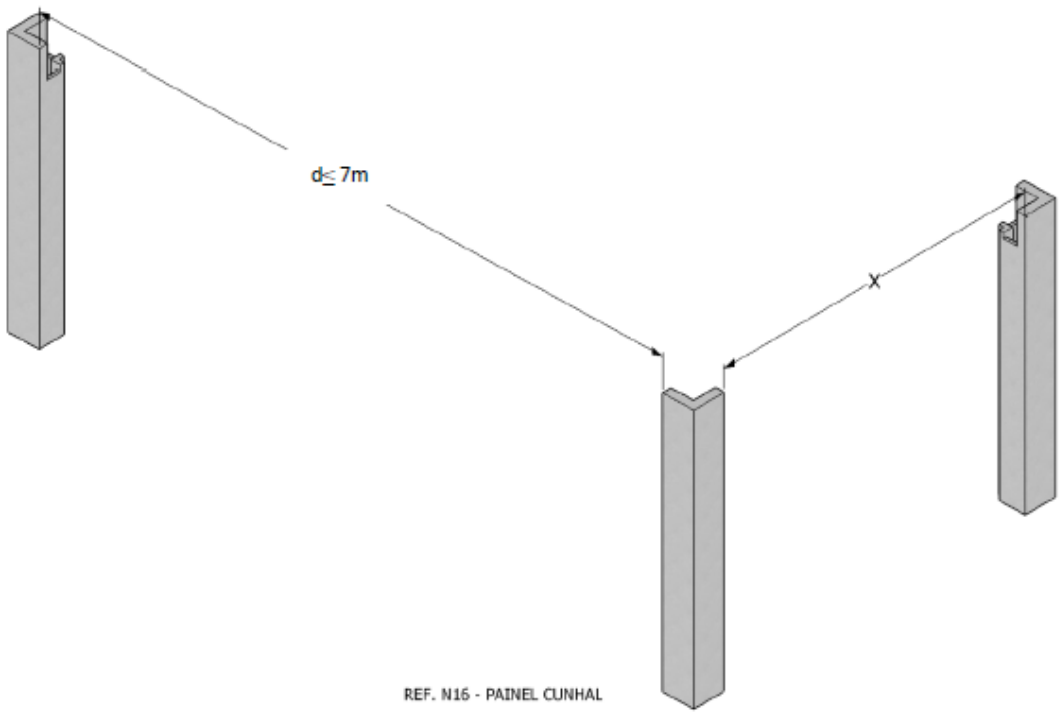

Figure 3.18: Representation of the max distance between wedges

Additional Structural Elements

In the case that the slabs have spans exceeding 7m in both directions, beams with a maximum length of 7m can be extended from the corner pieces to discharge on opposite walls.

3.6.2 Architectural Rules and Restrictions

These rules and restrictions are related to designing the building in a certain way following area specifications not breaking laws determined by the entities that regulate buildings.

Minimum Gross Area

All residences must have the following minimum gross floor area (measured by the outer limits of walls) per typology:

Table 3.6: Minimum area of the units by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Gross floor area in m^2	35	52	72	91	105	122	134	1.6*Ah

In the table 3.6, X represents the number of bedrooms and Ah is the livable area, which is the sum of the following areas, living room, kitchen, open space, and all the types of bedrooms.

Number and Type of Compartments

All residences must, by necessity, have a living room, kitchen, or living room/kitchen, and a complete bathroom.

Depending on their typology, the units should also have the following spaces:

Table 3.7: Number of bedrooms by housing typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Master Bedroom	-	1	1	1	1	1	1	1
Double Bedroom	-	-	1	2	2	3	3	x - 3
Single Bedroom	-	-	-	-	1	1	2	2

In the table 3.7, X represents the number of bedrooms.

Doors

The doors must have a clear width (measured free when open at 90°) of no less than 0.77m and a clear height of 2m.

In the case of doors where the jamb or adjacent wall is over 0.60m, their clear opening must have a width of 1.2m, which is the space things can go through when open.

Every door has a space on both sides that cannot be used by anything. This area's size is fixed according to the door type, as shown in Figure 3.19.

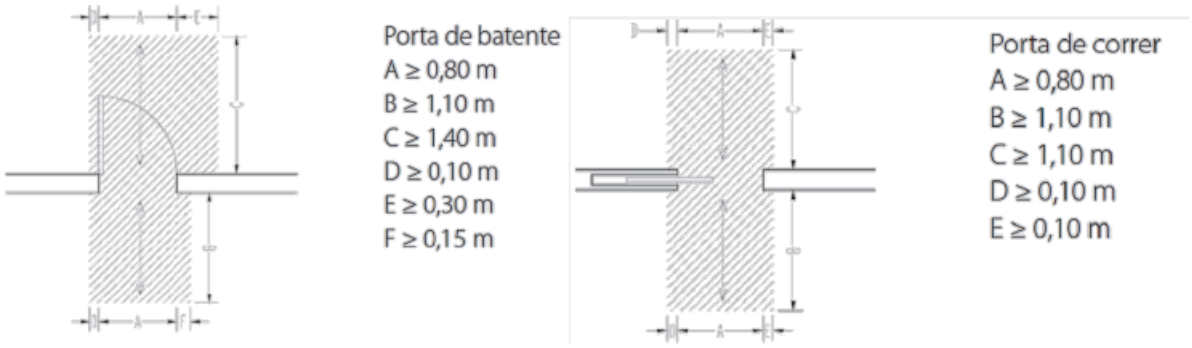

Figure 3.19: Representation of the dimensions of the free area doors must have

3.6.3 Types of Spaces and Associated Architectural Rules

Every construction must have a kitchen and a living room. The user must select what type of kitchen is desired, whether an open space or a separate kitchen and living room.

An extension must also be applied to the kitchen or the living room. The value of this extension depends on the total number of rooms.

Kitchens

The minimum area of the compartment (measured by the inner wall limit):

Table 3.8: Minimum kitchen area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	6	6	6	6	6	6	6	6

There should be a supplemental area in the living room or kitchen or living room/kitchen of at least:

Table 3.9: Minimum kitchen complement area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	6	4	6	8	8	8	10	number of bedrooms + 4

Minimum compartment size:

- When the area is less than 9.5 m^2, the minimum dimension between walls will be 2.10 m.

- When the area is greater than or equal to 9.5 m^2 and less than 12 m^2, a circle with a diameter of no less than 2.40 m should be inscribed in it.

- When the area is greater than or equal to 12 m^2 and less than 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it.

- When the area is greater than or equal to 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it. It is noted that the length may not exceed double the width, except when openings are applied to the two farthest opposite walls.

The compartment will always be illuminated and ventilated by one or more openings located in the walls in direct communication with the exterior. The total area of the opening(s) shall not be less than one-tenth of the compartment's area, with a minimum of 1.08 m2.

The minimum free distance between countertops or countertops and opposite walls must be at least 1.20 m.

In all kitchens, in the free space after the installation of countertops, it should be possible to inscribe a circle with a 1.5m diameter.

Living Room

The minimum area of the compartment (measured by the inner wall limit):

Table 3.10: Minimum living room area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	10	10	12	12	12	16	16	16

There should be a supplemental area in the living room or kitchen or living room/kitchen of at least:

Table 3.12: Minimum living room complement area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	6	4	6	8	8	8	10	number of bedrooms + 4

Minimum compartment size:

- When the area is less than 9.5 m^2, the minimum dimension between walls will be 2.10 m;
- When the area is greater than or equal to 9.5m^2 and less than 12 m^2, a circle with a diameter of no less than 2.40 m should be inscribed in it;
- When the area is greater than or equal to 12 m^2 and less than 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it;
- When the area is greater than or equal to 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it. It is noted that the length may not exceed double the width, except when openings are applied to the two farthest opposite walls.

The compartment will always be illuminated and ventilated by one or more openings located in the walls in direct communication with the exterior. The total area of the opening(s) shall not be less than one-tenth of the compartment's area, with a minimum of 1.08 m^2.

Living Room/Kitchen (Open Space)

The minimum area of the compartment (measured by the inner wall limit):

Table 3.14: Minimum open space area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	16	16	18	18	18	22	22	22

There should be a supplemental area in the living room or kitchen or living room/kitchen of at least:

Table 3.16: Minimum open space complement area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	6	4	6	8	8	8	10	number of bedrooms + 4

Minimum compartment size:

- When the area is less than 9.5 m^2, the minimum dimension between walls will be 2.10 m.
- When the area is greater than or equal to 9.5 m^2 and less than 12 m^2, a circle with a diameter of no less than 2.40 m should be inscribed in it.
- When the area is greater than or equal to 12 m^2 and less than 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it.
- When the area is greater than or equal to 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it. It is noted that the length may not exceed double the width, except when openings are applied to the two farthest opposite walls.

The compartment will always be illuminated and ventilated by one or more openings located in the walls in direct communication with the exterior. The total area of the

opening(s) shall not be less than one-tenth of the compartment's area, with a minimum of 1.08 m^2.

The minimum free distance between countertops or countertops and opposite walls must be at least 1.20 m.

In all kitchens, in the free space after the installation of countertops, it should be possible to inscribe a circle with a 1.5m diameter.

Laundry

The laundry room must have a minimum of $2m^2$ (measured by the inner wall limit).

Complete Sanitary Facility

Many rules must be followed in the bathroom design. There must be at least one full bathroom in the building, which includes a washbasin, toilet, bidet, and bathtub (with a minimum width of 0.80m). The minimum dimension for sanitary installations is $4.5m^2$.

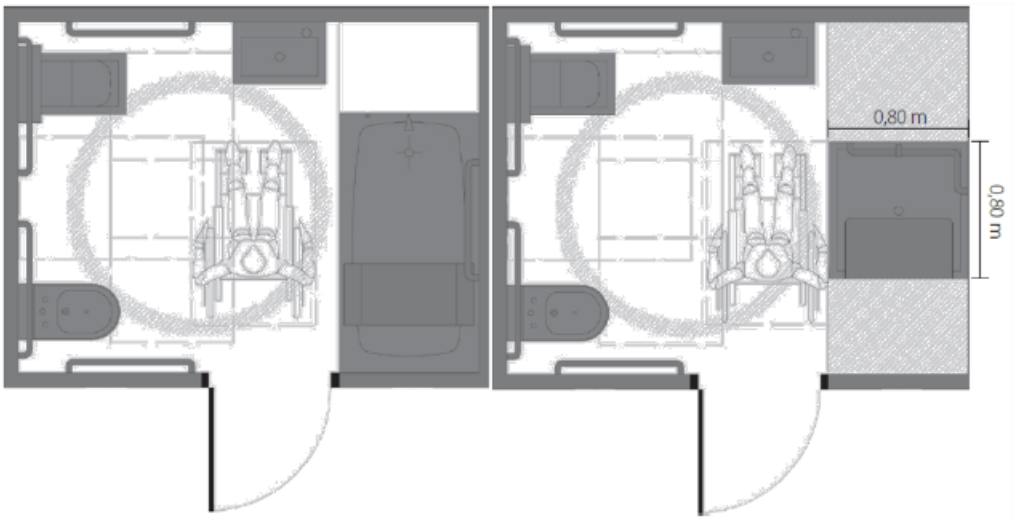

Figure 3.20: Complete bathroom representation

On one side of the toilet, a clear distance of 0.75m to another sanitary fixture or wall must be provided. In the free space after placing all sanitary fixtures, it should be possible

to inscribe a circle with a diameter of 1.5m, which does not overlap with the door swing area. Sliding or outward-opening swing doors are recommended in these spaces.

Sanitary Facility

Dwellings larger than T2 must have at least two sanitary installations with independent access. These sanitary installations do not need to be complete, but it is recommended to include a washbasin, toilet, and shower base with dimensions of 0.80x0.80m.

Master Bedroom

The minimum area of the compartment (measured by the inner wall limit):

Table 3.18: Minimum master bedroom area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	-	10.5	10.5	10.5	10.5	10.5	10.5	10.5

Minimum compartment size:

- When the area is less than 9.5 m^2, the minimum dimension between walls will be 2.10 m.
- When the area is greater than or equal to 9.5 m^2 and less than 12 m^2, a circle with a diameter of no less than 2.40 m should be inscribed in it.
- When the area is greater than or equal to 12 m^2 and less than 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it.
- When the area is greater than or equal to 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it. It is noted that the length may not exceed double the width, except when openings are applied to the two farthest opposite walls.

The compartment will always be illuminated and ventilated by one or more openings located in the walls in direct communication with the exterior. The total area of the opening(s) shall not be less than one-tenth of the compartment's area, with a minimum of 1.08 m^2.

Double Bedroom

The minimum area of the compartment (measured by the inner wall limit):

Table 3.20: Minimum double bedroom area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	-	-	9	9	9	9	9	9

Minimum compartment size:

- When the area is less than 9.5 m^2, the minimum dimension between walls will be 2.10 m.
- When the area is greater than or equal to 9.5 m^2 and less than 12 m^2, a circle with a diameter of no less than 2.40 m should be inscribed in it.
- When the area is greater than or equal to 12 m^2 and less than 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it.
- When the area is greater than or equal to 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it. It is noted that the length may not exceed double the width, except when openings are applied to the two farthest opposite walls.

Single Room

The minimum area of the compartment (measured by the inner wall limit):

Table 3.22: Minimum single bedroom area by typology.

Type of dwelling	T0	T1	T2	T3	T4	T5	T6	Tx>6
Area in m^2	-	-	-	-	6.5	6.5	6.5	9

Minimum compartment size:

- When the area is less than 9.5 m^2, the minimum dimension between walls will be 2.10 m.
- When the area is greater than or equal to 9.5 m^2 and less than 12 m^2, a circle with a diameter of no less than 2.40 m should be inscribed in it.
- When the area is greater than or equal to 12 m^2 and less than 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it.
- When the area is greater than or equal to 15 m^2, a circle with a diameter of no less than 2.70 m should be inscribed in it. It is noted that the length may not exceed double the width, except when openings are applied to the two farthest opposite walls.

Passageway

Passageways must have a minimum width of 1.2m.

Hall

At the entrances of dwellings (both inside and outside), it should be possible to inscribe a circle with a diameter of 1.5m that does not overlap with the door swing area.

Garage

It should have minimum dimensions of 3x5m. The garage door should be positioned in a way that it has at least 5m of free length in front of it.

3.7 Design Requirements

One of the restrictions of the system is that it should only be enable to draw lines with an angle of 90 (ninety) degrees between each other.

Other validations were added to the back end due to the impossibility of the frontend doing them.

Chapter 4

Development

This chapter delves into the implementation of code, highlighting the difficulties and limitations of the system. It also details the technical solutions to overcome the challenges encountered.

4.1 Development Technologies

FastAPI is a web framework for Python focused on building APIs. As its main characteristics, it is known because of being modern, fast, and simple, which means easy to use and learn. With these principles, it offers key features:

- Fast: Very high performance, on par with NodeJS and Go;
- Fast to code: Increase the speed to develop features by about 200% to 300%.
- Fewer bugs: Reduce about 40% of human (developer) induced errors.
- Intuitive: Great editor support. Completion everywhere. Less time debugging.
- Easy: Designed to be easy to use and learn. Less time reading docs.
- Short: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.

- Robust: Get production-ready code. With automatic interactive documentation.
- Standards-based: Based on (and fully compatible with) the open standards for APIs: OpenAPI and JSON Schema

These features can be found on its official website with documentation and benchmark tests to ensure the truth about the features (Ramírez, 2018).

The history of the framework explained by the creator is that the idea came when facing difficulties when using other frameworks, especially because many of them did not have support for the most recent Python resources as the type hint at the time of his research.

4.2 Configurations of the System

The stack of technologies used on the project was defined at the beginning by joining the most recent tools for Python web development.

Python, version 3.10, is a high-level, general-purpose programming language with an extensive standard library that covers many areas that can be used, becoming essential for the future challenges that could be found during the development. With it, the core library used for web development is FastAPI, version 0.96.0, a modern library that can bring freedom with high performance.

Other libraries used include Pydantic to create schemes for DTOs, version 1.10.8, Newer versions may cause problems not executing the system due to a file being removed for generic types. Numpy (1.25.0rc1) is also an essential component of the system because it offers many functions for numerical calculations responsible for reducing the time of implementing many functions for basic calculations. The libraries Python-Jose (3.3.0), Passlib (1.7.4), and Shortuuid (1.0.11) have implications on hashes used along the software, like passwords or IDs for components of the building. Version 2.0.1 of Shapely is an indispensable library that offers various functions for working with geometric shapes, such as areas, intersections, and functions to identify various aspects of a shape.

For the database, MongoDB, a NoSQL database that proposes to be faster to implement, highly scalable, and guarantees availability and integrity all the time (MongoDB, Inc., 2009). Its flexibility in creating data models was a point to consider as the platform in use at the front end did not have good documentation, so it was necessary to be able to change how the data was stored many times at the beginning of the development.

4.3 Common Functionalities

As the first tasks of the project, it was decided to organize the code conventions and to create structures that could be used along the development, such as common entities for the insertion on the database, enumerators for classification of objects on the system, the connection with the database was also done at the beginning.

The base entity to insert and retrieve information is defined to have a *created_at*, *updated_at* and a *deleted_at* field, with an extra field *id* when retrieving data.

```python
class BaseEntity(BaseModel):
    created_at: datetime = Field(default=datetime.now(tz=timezone.utc))
    updated_at: datetime = Field(default=datetime.now(tz=timezone.utc))
    deleted_at: datetime | None = Field(default=None)

    class Config:
        allow_population_by_field_name = True
        arbitrary_types_allowed = True
        json_encoders = {ObjectId: str}
```

Figure 4.1: Base Entity Class

Another data model used in endpoints is the response with pagination configuration. This model is used in most responses when retrieving a collection with many documents. It allows the front end to receive the total number of documents, the number of the next and the previous page, the maximum of documents returned for the page, the number of the current page, the number of the last page, and, at last, the documents properly.

```python
class PaginationResponseDto(GenericModel, Generic[T]):
    data: List[T]
    count: int
    next_page: Optional[int]
    prev_page: Optional[int]
    limit: int
    current_page: int
    last_page: Optional[int]

    class Config:
        arbitrary_types_allowed = True
        json_encoders = {ObjectId: str}
```

Figure 4.2: Pagination Response DTO

For the constants, there are the enumerators and the configuration for environmental variables. There are enumerators to classify the area type, the orientation of walls, hole types, object types, and the user role. These enumerators make the code cleaner to read, understand, and limit the input for these variables.

Only one decorator is used in this project, which is the one to get the pagination options on the request.

```python
class GetPagination(BaseModel):
    skip: int
    limit: int
    search: Dict | None
```

Figure 4.3: GetPagination Class

Classified as helpers, there are functions for hashes and functions related to the container loading system that need functions linked to artificial intelligence, such as fitness calculations, mutation, recombination, and selection.

Functions and routes related to logging in were also one of the first things done, as they are core components of the system. In this context, in the login route, it is verified if the user requested exists, then its password is verified. If everything is correct, a bearer token is generated to prevent the user from validating every request, and it is helpful to

verify the role in other routes.

```python
@router.post("/token", response_model=Token)
async def login_for_access_token(
    request: Request, form_data: OAuth2PasswordRequestForm = Depends()
):
    user = await authenticate_user(request, form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )
    response = Token(
        **user.dict(by_alias=True), access_token=access_token, token_type="bearer"
    )
    return response
```

Figure 4.4: Login function

4.4 Pre-processed data structure

The collection Graphs is a data structure that stores pre-processed data for validation. Its creation happens every time that the content of the building is saved. This process consists of a function that iterates over the values of the content and creates information about angles, areas, items, lines, corners, holes, beams, and others.

The first function it calls is the one to assign the angles between the walls of the building. This function goes through an iteration of all the vertices and then iterates through the lines to find the lines in which the vertex is present. The vertices adjacent to the one being iterated are stored in variable vertices. Later, it is done a combination of all of them is used to call another function that uses the math equation

$$< v, u >= cos\varphi.|v|.|u|$$

, which calculates the angle. In the end, the value in radians is saved on the graph structure in the vertex node.

```
def assign_angle(self, content, graph):
    for layer in content.layers:
        for vertice in layer.vertices.keys():
            vertices = []
            for line in layer.vertices[vertice].lines:
                v1 = layer.lines[line].vertices[0]
                v2 = layer.lines[line].vertices[1]
                if layer.vertices[v1].id == layer.vertices[vertice].id:
                    vertices.append({v2: layer.vertices[v2]})
                else:
                    vertices.append({v1: layer.vertices[v1]})
            vertices_combination = list(combinations(vertices, 2))
            for i in range(len(vertices_combination)):
                vertices_combination[i] += (
                    self.get_angle(
                        layer.vertices[vertice],
                        list(vertices_combination[i][0].values())[0],
                        list(vertices_combination[i][1].values())[0],
                    ),
                )
            graph.vertices[vertice] = VerticeNode(
                x=layer.vertices[vertice].x,
                y=layer.vertices[vertice].y,
                angles=[create_angle_model(x) for x in vertices_combination],
            )
    return graph
```

Figure 4.5: Function that assigns the angles

The assignment of the areas is also done using a math equation due to the necessity of calculating the area using only its vertices because the information about the area contained in the content of the building does not provide the walls it has, just the vertices.

The function iterates through the areas and its vertices to get the sum of the determinants of matrices composed by one line of the vertex in the iteration and another one that is this vertex plus one. In the end, the absolute value of this sum is divided by two.

```python
def assign_area_value(self, content, graph: Graphs):
    for layer in content.layers:
        for area in layer.areas.keys():
            determinant = 0
            for i in range(len(layer.areas[area].vertices)):
                j = (i + 1) % len(layer.areas[area].vertices)
                determinant += np.linalg.det(
                    [
                        [
                            layer.vertices[layer.areas[area].vertices[i]].x,
                            layer.vertices[layer.areas[area].vertices[i]].y,
                        ],
                        [
                            layer.vertices[layer.areas[area].vertices[j]].x,
                            layer.vertices[layer.areas[area].vertices[j]].y,
                        ],
                    ]
                )
            graph.areas[area] = AreaNode(
                value=abs(determinant) / 2,
                areaType=layer.areas[area].properties["areaType"],
                name=layer.areas[area].properties["name"],
                vertices=layer.areas[area].vertices,
                internal_value=None,
                lines=[],
                items=[],
            )
    return graph
```

Figure 4.6: Function that assigns the area value

To assign the lines to the areas, a function iterates through lines and areas, verifying if the vertices of the line are present on the vertices of the area, and then it is assigned to the area node.

The logic used to create the function to identify external walls (lines) is to iterate the areas (A1) and then its lines. Iterate these lines through all the areas (A2), and if at the end of the iteration of A2, the line is only present in A1, it is because it is an external wall.

```python
def assign_external_lines(self, graph: Graphs):
    for area_1 in graph.areas.keys():
        for line in graph.areas[area_1].lines:
            internal = 0
            for area_2 in graph.areas.keys():
                if area_1 != area_2 and line in graph.areas[area_2].lines:
                    internal += 1
            graph.lines[line].external = True if not internal else False
    return graph
```

Figure 4.7: Function that assigns the external walls

The value of the internal area of the building was done by iterating the areas and subtracting from the value already assigned to the area of the walls.

```python
def assign_internal_area(self, graph: Graphs):
    for area in graph.areas.keys():
        squares = []
        vertices = []
        prev_x = -math.inf
        prev_y = -math.inf
        for line in graph.areas[area].lines:
            if (
                graph.lines[line].vertices[1][0] != prev_x
                and graph.lines[line].vertices[1][0] != prev_y
            ):
                vertices.append([graph.lines[line].vertices[0]])
                prev_x = graph.lines[line].vertices[0][0]
                prev_y = graph.lines[line].vertices[0][1]
            squares.append(graph.lines[line].length * (graph.lines[line].thickness / 2))

        diff = ((len(vertices) - 2) / 2) - 1

        squares.append(
            -((graph.lines[line].thickness / 2) * (graph.lines[line].thickness / 2))
            * (len(vertices) - diff)
        )
        squares.append(
            ((graph.lines[line].thickness / 2) * (graph.lines[line].thickness / 2)) * diff
        )
        graph.areas[area].internal_value = graph.areas[area].value - sum(squares)
    return graph
```

Figure 4.8: Function that assigns the internal value for the areas

The items are assigned differentiating between accessibility and ordinary items. If it

is an accessibility item, the node is created simply by adding a circle's radius and center. On the other hand, if it is a different item, it is necessary to calculate the vertices of it.

The items are assigned to areas by creating polygons of the areas and verifying if the initial vertex of the item is inside it. This was done to avoid generating ambiguous information if the user put an item inside a wall intersecting two areas.

The assignment of lines is straightforward. It iterates through the lines in the content and calculates its length and the adjacent lines by iterating through all the other lines. Before inserting in the graph, it is verified its orientation, if it is a horizontal or a vertical line.

```
def assign_lines(self, content, graph: Graphs):
    for layer in content.layers:
        for line in layer.lines.keys():
            x1 = layer.vertices[layer.lines[line].vertices[0]].x
            y1 = layer.vertices[layer.lines[line].vertices[0]].y
            x2 = layer.vertices[layer.lines[line].vertices[1]].x
            y2 = layer.vertices[layer.lines[line].vertices[1]].y
            length = math.dist((x1, y1), (x2, y2))
            lines_adj = []
            for line_2 in layer.lines.keys():
                if line == line_2:
                    continue
                line_2_obj = layer.lines[line_2]
                if (
                    layer.lines[line].vertices[0] in line_2_obj.vertices
                    or layer.lines[line].vertices[1] in line_2_obj.vertices
                ):
                    lines_adj.append(line_2_obj.id)
            if x1 == x2:
                graph.lines[line] = LineNode(
                    direction=DirectionType.VERTICAL,
                    vertices=[(x1, y1), (x2, y2)],
                    vertices_ref=layer.lines[line].vertices,
                    thickness=layer.lines[line].properties["thickness"]["length"],
                    length=length,
                    external=False,
                    holes=layer.lines[line].holes,
                    is_invisible=True
                    if layer.lines[line].type == "invisible wall"
                    else False,
                    lines_adj=lines_adj,
                )
            elif y1 == y2: ···
            else: ···
    return graph
```

Figure 4.9: Function that assigns lines to the graph

The holes are another type of node that needs more calculations than the others because they require an empty area around doors and store the vertices of windows and doors. For this, it is done an iteration in the holes of the content, and there is a condition for each type of hole, standard door, sliding door, door of a module, and window, to add the correct value to each node.

```
def assign_holes(self, content, graph: Graphs):
    for layer in content.layers:
        for hole in layer.holes.keys():
            hole_obj = layer.holes[hole]
            if hole_obj.obj_type == "door" and hole_obj.type == "sliding door": ···
            if hole_obj.obj_type == "door" and hole_obj.type == "door":
                free_area = []
                obj_area = []
                size = hole_obj.properties["width"]["length"]
                if graph.lines[hole_obj.line].direction == DirectionType.VERTICAL: ···
                elif graph.lines[hole_obj.line].direction == DirectionType.HORIZONTAL: ···
                graph.holes[hole] = HoleNode(
                    size=size,
                    hole_type=HoleType.INTERNAL_DOOR,
                    direction=graph.lines[hole_obj.line].direction,
                    free_area=free_area,
                    object_area=obj_area,
                    vertex=hole_obj.vertex,
                    height=hole_obj.properties["height"]["length"],
                    thickness=hole_obj.properties["thickness"]["length"],
                )
            elif hole_obj.obj_type == "door": ···
            elif layer.holes[hole].obj_type == "window": ···
    return graph
```

Figure 4.10: Function that assigns holes to the graph

Two of the last functions in creating the graph calls are related to the beams, where they will be assigned and find the best combination for the building.

The *assign_beams* function generates all the possibilities of where beams can be applied in the building. It is done by iterating the vertices on the graph and verifying if they are corners. If this is true, it is tested whether adding a beam of the max value it can support or smaller is possible. With all these validations returning true, a beam is created on the graph with its beginning, ending, orientation, and length.

The following function finds the best beams. All possible combinations are produced using the beams already added to the graph. The building's external vertices are located without vertices in the middle of straight lines. Once all of these details are in place, the function begins to verify the feasibility of beams.

The iteration of the possibilities starts by adding vertices of the beams not included in the external vertices. Then, it creates the areas generated by adding the beam. In this

case, as it is tested with one beam per iteration, only two areas are created. After all, it is necessary to get the intersections of the areas, so two by two areas are being tested to find the intersection, and new areas are being created. It ends when no more intersections exist. It is then possible to test whether the wall and beams can support the building.

In order to facilitate the validation, a line node for the vertices and beams is created as if everything was a wall. With the information about the lines of the areas, the distance between the lines is tested, and if the beam ends in the middle of a window or door. The best option will be obtained from the combinations that are acceptable to construct the building and that use less length of beams.

4.5 System Rules

The functions that solve the system's rules were written in graph services, as most of them validate the information stored on the graph of the building. If the validation goes wrong in any of the functions, a message explaining the error is returned.

4.5.1 Structural Rules and Restrictions

The rules of this subsection have its calculations realized during the creation of the pre-processed data structure.

Clear Span of Slabs and Additional Structural Elements

The validation of the possibility of construction is available on the graph as the calculation to find the best configuration for the beams was already done. Then in the validation process, it is only verified if the property responsible for saving the possibility of construction is true or not.

4.5.2 Architectural Rules and Restrictions

These rules are the core of the validation process, even without the same impact as the structural rules, any of the buildings can be constructed without passing them.

Minimum Gross Area

The external area of the building, which is calculated using the same system as the internal area, is used as the total area of the building; however, it sums rather than subtracting the area of the walls. Then, for a building with more than six rooms, the product owner defined a multiplier of 1.6 on the chosen areas.

Number and Type of Compartments

The total number of rooms is validated along with the number of each type of room, and a message is returned if the total number of rooms differs from the topology of the building as shown. This is done with the previously calculated values that were reached by iterating over the areas and counting the number of rooms depending on the type of area selected by the user.

Doors

The validation of the clear width and minimum height is done by iterating over the holes verifying the property height and calculating width when open with a different approach for external doors and internal doors, which for the external door it is important to disconsider the module component on both sides of the door.

The validation of the free area in the door's space is done using the library shapely to create a polygon for the door's area and one for the tested item. If they intersect, the rule fails, and a message with the item's name followed by its coordinates will be returned for the user to fix its location.

4.5.3 Types of Spaces and Associated Architectural Rules

Depending on the type, the area is validated by looking for the more extensive area among the areas of the chosen type, and if it corresponds to a value higher than the rule requires, the validation passes.

Kitchens, Living Rooms, Open Space and Bedrooms

Validating the minimum area is just an iteration over the areas checking the internal area value of them. The biggest problem here is to validate the minimum dimensions of them. For that, it is in use a system to get the walls of the area in pairs and then verify the distance.

The system used to find walls in the same orientation is to get a pair of walls and verify the orientation and if more than a pair are in the same orientation, it is considered the one with the highest distance. Considering the area, a minimal distance between parallel walls or an accessibility item with a specific radius is required. With an area smaller than $9.5m^2$, the minimum distance between the wall is 2.10 meters, the other area values where the rule changes are smaller than $12m^2$, smaller than $15m^2$ and then bigger than $15m^2$. For these last values, it is necessary to include an accessibility item with sizes 1.20, 1.35, and 1.35 meters of radius, respectively. In the same function, it is validated whether there is an accessibility item of 0.75m radius in the bathroom, but it does not depend on the area.

Laundry

Using the internal value property present on the areas node in the graph, the function tries to find a laundry that satisfies the rule. If there is not, an error message is returned.

Complete Sanitary Facility

The function that validates if there is a full bathroom in the building, loops through the bathroom items, and if there is a bathroom with every item, the validation succeeds.

The distance of walls and items to the toilet is validated using polygons and line strings in other to check the distance between them. Iterating over all the items present in the area and also the walls that are part of it.

The validation of the accessibility item is the same for all the areas, it is verified if it is present in the area, and the function that verifies the free area of the door also checks for accessibility items.

Sanitary Facility

This rule is checked simply by verifying if the building is a T2, and then it is verified if there is more than one sanitary facility in it.

Passageway

The validation of the hallways has two things that can lead to problems: the user must draw its start and end positions with invisible walls/lines and mark the area as a passageway or mark the checkbox that the building has a passageway. Otherwise, it will not work. There is an iteration that looks for areas marked as passageways. Finding one, it gets its physical walls and calculates the distance between them. It passes if it is wider than 120cm (centimeters).

Hall

The hall area validation has the same problem as the hallways validation due to the difficulty of identifying these areas using simple points in a Cartesian plane.

The validation of the hall is related to the accessibility objects. So the validation is to verify if the area has one accessibility item of 75cm radius.

Garage

The garage validation follows almost the same strategy used to validate wall distance in the kitchen, bedrooms, and living room. The rule says that the garage must have a

dimension of at least 3x5m, with the gate being on the wall of 3m. So it is necessary to find the distance between the walls using the process of finding a pair of walls with the same orientation and then checking the higher distance.

4.6 Design Requirements

At the beginning of the project, the limitation of creating walls with only 90 degrees was not working on the front end, so there is a function to verify this rule that can affect the construction later if not fixed.

After the angles defined in the graph, the function iterates over all the values and verifies if it corresponds to 90 degrees.

```
def validate_building_angles(self, graph: Graphs):
    for vertice in graph.vertices.keys():
        vertice_obj = graph.vertices[vertice]
        for angle in vertice_obj.angles:
            if angle[2] != math.pi / 2 and angle[2] != math.pi:
                return [(f"All the angles of the building must be 90 degrees", False)]
    return []
```

Figure 4.11: Function that iterates over the vertices

Other validations were added to the back end due to the impossibility of the frontend doing them. These validations include verifying items overlapping walls or other items, vertices over holes, and holes overlapping holes.

4.7 Container Loading

The source code used as the fundamental basis for this service was initially developed for the project "A hybrid multi-objective genetic algorithm for the container loading problem" by Nivedha Ramesh and is publicly available on the GitHub repository (https://github.com/Nivedha-Ramesh/Container-Loading-Problem). Some modifications were made to simplify its operation and meet the software requirements proposed in this work.

The dataset was completely modified to be used as input with the necessary information about the components. It contains the reference number of each component and the number of pieces that are packaged together. The information about the boxes, container size, and total amount of boxes is stored in a JSON object.

The container load function was split into two functions, one for getting the coordinates of each package and another for generating the information used in the image that the front end presents.

Using the information about the boxes and the container dimensions, the packages are distributed many times through several generations, performing ranking, crossover, mutation, and selection through the generations.

The fitness function uses the simple deepest bottom left with fill (S-DBLF) algorithm as a heuristic to pack the boxes in the container. It calculates the utilization space, the number of boxes packed, and the total value of the boxes packed into the container as the fitness values. It moves each box to the deepest available position to achieve these results, then as much to the bottom, and as much to the left as possible Ramesh and Umashankar, 2021.

The rank function is made using the NSGA II (nondominated sorting genetic algorithm), an evolution of the simple sorting genetic algorithm (SGA) with better time complexity. In simple SGA, the time complexity is O(MN3), where M is the number of objectives, and N is the size of the dataset. NSGA II has a time complexity of O(MN2). The function ranks the population into different fronts and calculates the crowding distance according to NSGA II Ramesh and Umashankar, 2021.

An order-based crossover is used due to its adequacy for permutation encoding. Two other parents produce two child individuals, and then two points are chosen to create the portions for the multi-crossover. The middle of the split of one parent is copied for only one child, and then the split of the other parent is copied to the other child. The other values come from a swept circulation of the remaining values of the parents respectively Ramesh and Umashankar, 2021. The crossover is exemplified in Figure 4.12.

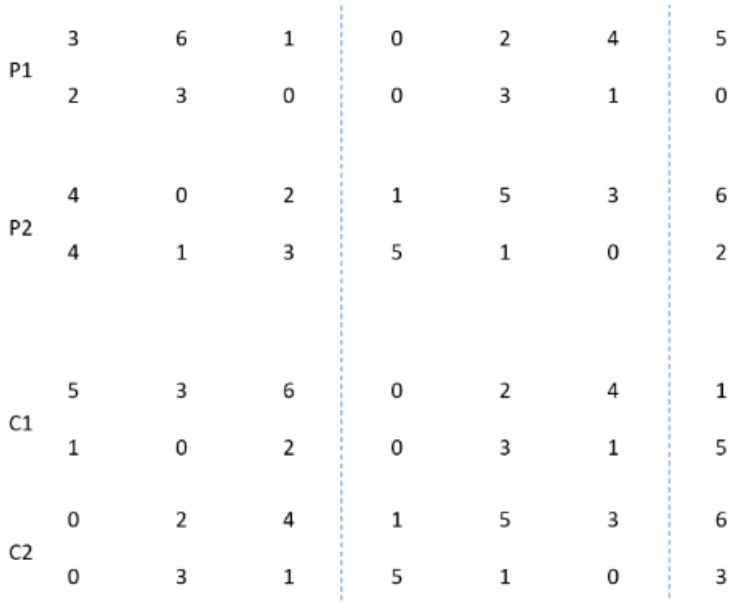

Figure 4.12: Example of modified order crossover. Reprinted from "A hybrid multi-objective genetic algorithm for the container loading problem" by N. Ramesh, 2021, p. 4

For the mutation, two locations on the chromosomes are chosen where the middle of the split will be reversed. Later, there is a possibility that the bits of the second chromosome will change to a different rotation value Ramesh and Umashankar, 2021. The mutation process is exemplified in Figure 4.13.

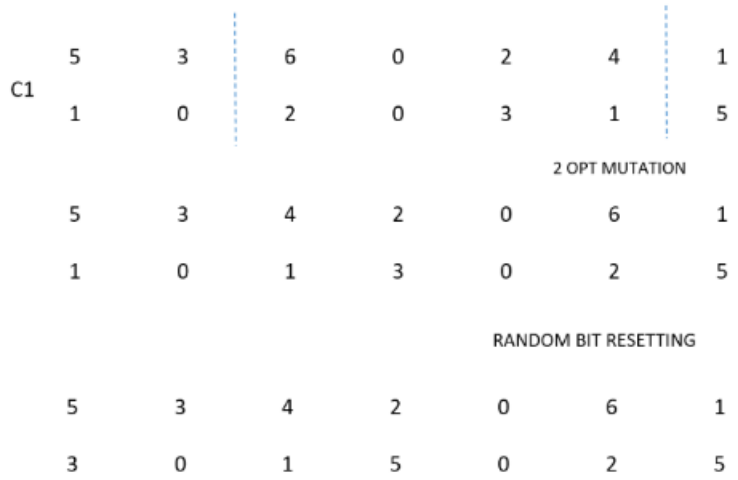

Figure 4.13: Example of two-step mutation.Reprinted from "A hybrid multi-objective genetic algorithm for the container loading problem" by N. Ramesh, 2021, p. 5

The initial population and the crossover and mutation child populations are used in the selection process, and the packing algorithm assesses the fitness of the individuals. They are then ranked and divided into various non-dominated fronts before being selected to make up the new parent population. Until a population equal to the prior parent population is reached, the individuals are chosen based on the best-ranked individuals (Ramesh & Umashankar, 2021).

Only the first individual marked with the highest rank is ultimately chosen to be presented to the user. The distribution function is called repeatedly until all the packages have been placed in all the required containers if not all of them can fit in a single container.

Once all the containers have been defined, it is possible to create the meshes the user will see on the 3D graph represented by Figure 4.14.

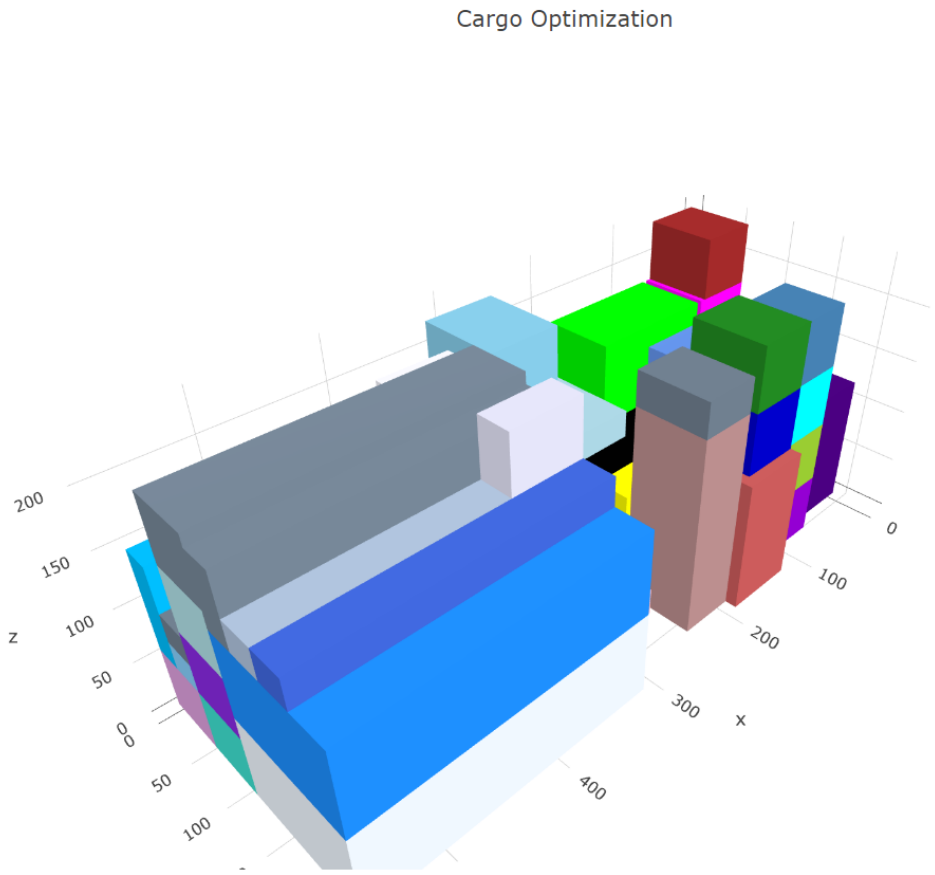

Figure 4.14: Example of cargo optimization

Chapter 5

Tests/Discussion

This chapter discusses the tests conducted by the project owner with professionals who tried the software and suggested improvements during the development.

5.1 Architects tests

Following the agile methodology adopted at the beginning of the project, tests were done on every new release. Then, bug reports and suggestions were received by email and discussed during the meetings. The suggestions were not met as expected on every occasion. Instead, some expectations were met through alternative approaches. The bugs and suggestions for the back-end development will be shown in a table for better comprehension.

Table 5.1: Bugs related in November 2022

Description	Date
When testing the model, I noticed that the software is allowing the drawing of walls with angles other than 90 degrees. When I instruct the system to correct the angles, it doesn't make any changes.	11/Nov/2022
While creating a house model, I observed that the software does not recognize the house type based on the number of bedrooms. Additionally, after validation, it identifies several non-conformities that are not present in the model I designed, such as the kitchen having less than 6 square meters.	11/Nov/2022

The first bug used to happen more when the walls could not be drawn only following the X and Y axis due to a division per zero that could happen depending on the direction in which the lines were drawn. Now, with this feature implemented, the function to fix the angles is not necessary, and also, with the lines only being able to be drawn on the X and Y axis, this bug does not appear anymore.

The second bug was a problem with the enumerator of area types, in which when comparing the area type, instead of using the name of the enumerator, it was using the number provided on the request. After making this change, it worked fine.

On 23^{rd} December of 2022, the bug linked to the back end was more a feature request than a bug because the problem was that it was possible to put holes overlapping holes. It was not a feature that could be implemented on the front end at first because the project does not have a collision module, so it was decided to be a back-end task to verify if any door or window was overlapping one another. It was solved by mapping the vertices of the project's windows and doors and then iterating over them to find an intersection on these points.

On 22^{nd} February of 2023, there was only one bug: interior walls could be placed in

the middle of doors or windows. To solve this problem, an iteration over the vertices and the area of the holes looking for an intersection of them returns a message with the coordinates of the vertex that is intersecting for the user to solve the intersection.

Table 5.3: Bugs related in April 2023

Description	Date
Verification that all compartments have access doors or openings.	19/Apr/2023
Ensure that the doors (including their swing), do not overlap with walls, furniture, and other elements.	19/Apr/2023
When there is an overlap between the accessibility circle and the components, specify in which compartment it occurs.	19/Apr/2023

The problem of the areas having to have at least one door was solved by verifying the holes and analyzing if at least one is a door. It can be for the inside or outside areas.

The problem of the door overlapping items happened due to bad communication at the beginning, where the example image shows the swing area of the door overlapping an accessibility item. However, it is easy to fix later by simply adding the validation using the intersection method previously used in other functions, but this time for the accessibility items. For other items, it was already working. By creating this part of the code, the last bug was also solved by adding the name of the area where the accessibility item is if there is any problem.

Table 5.5: Bugs related in May 2023

Description	Date
In living rooms/kitchens, it is mandatory to have a window with specific dimensions. In the example of the attached floor plan, the software assumes that the entrance door serves as the required lighting/ventilation opening as per the law. The entrance door cannot be considered for this purpose.	03/May/2023
In complete bathrooms, the accessibility circle cannot overlap with the door opening. Is this considered in the program? Additionally, the accessibility circle can overlap with the shower base.	03/May/2023
Whenever I validate the project, it claims that the accessibility circles overlap with the doors. However, this was not happening.	25/May/2023
Regarding the structural rules, the software only states that construction is not possible. It should provide an explanation of why.	25/May/2023

The first bug of 3^{rd} of May happened due to the lousy communication about the rule. At first, the doors could be considered as an opening for ventilation. To change this behavior, a simple condition on the function to verify the openings in the areas and consider only holes of the type window.

The second report cannot be considered a bug because the client never provided this information, and the example sent did not follow this rule. To apply it, the verification of items overlapping the door opening area now has to verify accessibility items but does not consider the shower base.

The first bug of 25^{th} of May was related to internal doors in which the size used for

the free area they must have was equal to the values for external doors. By changing it, the behavior got normal.

The second comment about the software was about changing the message for when the building cannot be constructed. The message changed to "The maximum distance between parallel exterior walls must be less than 6/7m. The max size for the windows is 3 meters in each wall", which has a better explanation for the user of how he can fix his building.

Table 5.7: Bugs related in June 2023

Description	Date
The software indicates that each compartment needs to have an interior door, even when one already exists.	21/Jun/2023
When there is an issue with the space for a sanitary fixture, it would be helpful to know which one. For example, if there are multiple bathrooms and one of the toilets requires more space, it would be beneficial to be able to identify that specific toilet.	21/Jun/2023
It is valid to have either a bathtub or shower base in bathrooms. Currently, the program only indicates that everything is okay if there is a bathtub.	21/Jun/2023

The bugs presented this month were all simple mistakes during development. The first was about forgetting to verify both types of doors in the areas, not only external ones. The second is not a bug but a request for changing the message of space on the sides of the toilet, which requires a minimum space on one of its sides. The modification was to put the name of the bathroom in the message, but it requires that the user put a name on it.

The last one was also a simple mistake of forgetting to validate the shower base and not only the bathtub.

Table 5.9: Bugs related in July 2023

Description	Date
In the validation process, the software still generates an error if the total sum of openings in one wall exceeds 3m. As I mentioned in our recent meetings, this limitation no longer applies. Now, only the distance between walls matters. Regarding the distance between walls, please only reference 7m and not 6/7m.	17/Jul/2023
It shows an error stating that the accessibility circle is missing in the hall, even though it is already present.	17/Jul/2023

After testing the materials, the product owner brought new info that it was not necessary anymore to limit the size of holes in a wall. With this change, the verification of the size of holes while validating the possibility of constructing was removed.

The bug with the accessibility items on the hall happened because of a duplicated validation of the accessibility items. The hall has a validation for it, and there is a function to validate accessibility items over the doors' free area. The validation on the accessibility items over doors was removed, and now it validates if the accessibility item overlaps the free area of doors, and the hall validation was changed to consider any accessibility items on the hall.

Chapter 6

Conclusions

The work presented consisted of developing a system that would help architects who work with buildings made of pre-fabricated specific components, including an area for management of the validation and delivery of the components necessary for the project without using excessive resources.

The platform for drawing the building already has most of the necessary components to design a house, just in need of more furniture for a better user experience using the software. Including many functions to meet the requirements and pass the validations, the software is ready for the final user to create reliable buildings.

The system created to manage the orders with the intelligent logistic system provides a good interface for the company to inform the users about their orders and to get an optimized load for the cargo of the building inform the people responsible for the delivery of the best way to organize and reduce costs of the shipping process.

The work also aimed to reach a standard user who wants to build his own house, achieving the minimum requirements for it to be built by an architect.

During the development, knowledge of several areas was necessary as it is a multidisciplinary project, and also, because of the format, much information was given. Many geometry calculations were required, especially at the beginning when the libraries for optimizing code were not known. The biggest challenge was understanding all the areas involved in the project, such as civil construction.

The code structure was intentionally designed to ease maintenance in the coming years, keeping up with the latest technology stacks. Incorporating new features in the future should pose no issues.

6.1 Future Work

Until this point, the platform is working as expected and has already been validated by the stakeholders. However, improvements in performance and security are welcome, not only for the present features but also for new features that might be included.

The following steps for the back end include the development of new validations that could not be developed in this first stage, such as the validation of kitchen cabinets that still need to be done. Another validation necessary in the future is the one related to stairs for a second floor, as having two floors was not a priority.

A good feature for study would be to make the rules changeable and not hard-coded for the back end to be able to be adapted for other systems and not only for a single purpose. Many of the features are working for this to be possible in the future, as the modules and the catalog are already out of the code, except for the part on cargo optimization.

Regarding the code, the database connection is adequate, but an alternative approach may be required if scalability becomes a priority. The rollback functionality is also necessary in some cases, and for now, it is being done in a very primitive way.

Bibliography

AMA, A. P. A. M. A. (2023, January 31). *Micado .: Micado - modular insulated concrete core - advanced and optimized panelized production system*. Retrieved March 15, 2023, from https://transparencia.gov.pt/pt/fundos-europeus/beneficiarios-projetos/projeto/NORTE-01-0247-FEDER-113482

Autodesk. (2002). *Revit* (2022). Retrieved April 26, 2023, from https://www.autodesk.com/products/revit

Barsa, H. (2021, October 4). *O sketchup é bim! e você ainda não sabe?* Retrieved April 6, 2023, from https://gabster.com.br/arquitetura/sketchup-e-bim/

BIMTech. (2016, September 22). *What is bim and how it came to be*. Retrieved March 15, 2023, from https://bimtech.eu/bim/

CeDRI, R., & IPB, P. (2021). *Research centre in digitalization and intelligent robotics*. Retrieved April 26, 2023, from https://cedri.ipb.pt/

[CVDLAB], R. (2019, November 3). *React planner*. Retrieved April 4, 2023, from https://cvdlab.github.io/react-planner/

Easyhome New Retail Group & Alibaba Group. (2009). *Home design - homestyler*. Retrieved April 6, 2023, from https://www.homestyler.com/about/story

Farias, J. C. (2019, December 2). *O que é o revit?* Retrieved April 6, 2023, from https://spbim.com.br/o-que-e-o-revit/

F.F., A., Akbar Nezhad, A., Rashidi, T., & Waller, S. (2014). Importance of planning for the transport stage in procurement of construction materials, 8. https://doi.org/10.22260/ISARC2014/0062

Haas, C., Fagerlund, W., University of Texas at Austin. Department of Civil Engineering. PPMOF Research Team, P. 1., & of Texas at Austin. Construction Industry Institute, U. (2002, January 9). *Preliminary research on prefabrication, pre-assembly, modularization and off-site fabrication in construction*. Construction Industry Institute. https://www.construction-institute.org/resources/knowledgebase/knowledge-areas/modularization/topics/rt-171/pubs/rr171-11

ISO, S. (2016). 29481-1:2016 (e) building information modelling—information delivery manual—part 1: Methodology and format.

Jorge Evaristo Cuntín Rey, José Luis Mosquera Senande, Margarita Novo Malvárez, Yolanda Pardellas Márquez, José Alfredo Garrido Núñez, Jesús M. González Pérez, Ana María Rodríguez Moreira, Teresa García Durán, José Manuel Malheiro Gutiérrez, Helio David Abreu Da Silva, & Sonia Heleno Rodríguez. (1999). *Eures transfronteirizo galicia-norte de portugal*. IFES. Retrieved October 30, 2023, from https://www.eures-norteportugal-galicia.org/wp-content/themes/eures/contenidos/cds/cd03/p/001.html

Lu, N., & Korman, T. (2010). Implementation of building information modeling (BIM) in modular construction: Benefits and challenges. *Proceedings of the 2010 ASCE Construction Research Congress*, 1136–1145. https://doi.org/10.1061/41109(373)114

Martello, S., Pisinger, D., & Vigo, D. (1998). The three-dimensional bin packing problem. *Operations Research*, *48*. https://doi.org/10.1287/opre.48.2.256.12386

MongoDB, Inc. (2009). *Mongodb* (2023). Retrieved August 28, 2023, from https://www.mongodb.com/

Mónica Silvares. (2021). *Portugal é o segundo país europeu com maior rede de autoestradas por habitante*. ECO. Retrieved October 26, 2023, from https://eco.sapo.pt/2021/08/12/portugal-e-o-segundo-pais-europeu-com-maior-rede-de-autoestradas-por-habitante/

Navarro, A. F. (2012). Acidentes causados durante a movimentação de cargas, 26.

Ramesh, N., & Umashankar, J. R. (2021, January 19). *A hybrid multi-objective genetic algorithm for the container loading problem*. Retrieved April 11, 2023, from https://github.com/Nivedha-Ramesh/Container-Loading-Problem

Ramírez, S. (2018). *Fastapi* (2023). Retrieved October 27, 2023, from https://fastapi.tiangolo.com

Razor. (2019, October 29). *Sketchup: O que é e qual sua contribuição para arquitetura?* Retrieved April 6, 2023, from https://razor.com.br/blog/tecnologia/sketchup/

Sun, C., Jiang, S., Skibniewski, M. J., Man, Q., & Shen, L. (2017). A literature review of the factors limiting the application of bim in the construction industry. *Technological and Economic Development of Economy*, *23*(5), 764–779. https://doi.org/10.3846/20294913.2015.1087071

Trimble Inc. (2023). *Sketchup* (2023). Retrieved April 26, 2023, from https://www.sketchup.com/

Volk, R., Stengel, J., & Schultmann, F. (2014). Building information modeling (BIM) for existing buildings — literature review and future needs. *Automation in Construction*, *38*, 109–127. https://doi.org/https://doi.org/10.1016/j.autcon.2013.10.023