



**Politecnico
di Torino**

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Edile

A.a. 2023/2024

Sessione di Laurea Ottobre 2024

Web-Based IFC Application for Infrastructure Models: Enhancing User Interaction Through Natural Language Processing

Supervisor:

Prof. Anna Osello

Co-Supervisor:

Ing. Nicola Rimella

Arch. Elisa Stradiotto

Candidate:

Ervin Alla

Acknowledgment

"I am deeply grateful to God for granting me the strength and guidance to achieve this goal, and to my family for their unwavering support.

I would like to thank the supervisor of my thesis, Prof. Anna Osello, and especially the co-supervisors, Nicola and Elisa, for their invaluable advice and support throughout this process.

I would also like to extend my thanks to Webuild S.p.A., for generously providing the models and files that were essential in conducting this research."

Abstract

Building Information Modeling (BIM) has revolutionized the Architecture, Engineering, Construction, and Operations (AECO) industry by providing a more collaborative approach to planning, designing, and managing building projects. Among the various standards used in BIM, Industry Foundation Classes (IFC) stand out as a vital open BIM standard by promoting interoperability across different software applications and enabling continuous data exchange throughout a project's lifecycle. As infrastructure projects grow in complexity, more sophisticated tools are being developed to improve model visualization and user interaction, especially in web-based environments.

This thesis addresses this need by developing and evaluating a web-based IFC viewer adapted for infrastructure models. By leveraging cutting-edge open-source JavaScript libraries and Natural Language Processing (NLP) technologies, the proposed solution enhances visualization and user interaction within BIM workflows. The innovative tool offers synchronized views of model alignments, profiles, and cross-sectional visualizations at specific points along the 3D alignments. As a result, it provides users with a more intuitive and broad understanding of the model.

A distinctive feature of the application is an integrated Conversational Agent which allows users to interact with the model in a more accessible and user-friendly approach. This functionality is delivered through a web interface, eliminating the need for installed software and local processing power, as a result making it accessible from any device with an internet connection.

A real-world case study is presented which introduces a workflow that starts with a BIM authoring software model, extends to adding alignments, and concludes with exporting the models to IFC format for web visualization. This workflow was used to demonstrate a practical application which can be replicated with other models.

The model is then used to test the IFC Viewer and the integrated Conversational Agent, analyzing the advantages and the drawbacks that arise when using these tools. The results describe the innovative solutions this viewer brings and also the potential future improvements that can take it a step further.

Accessing advanced visualization with a user-friendly interface can improve how we manage and interact with digital infrastructure models. The developed IFC viewer makes it easier to work with complex infraBIM data, but it also introduces innovative tools that can be further explored in future research within the AECO industry.

Keywords

openBIM, Web application, Natural Language Processing, Chatbot, InfraBIM, Open Source

Abbreviations/Acronyms

BIM – Building Information Modeling
IFC – Industry Foundation Classes
NLP – Natural Language Processing
AECO – Architecture, Engineering, Construction and Operations
ISO – International Organization for Standardization
API – Application Programming Interface
FHWA – Federal Highway Administration
CAD – Computer-Aided Design
DXF – Drawing Exchange Format
UI – User Interface
CX – Customer Experience
NLU – Natural Language Understanding
IC – Intent Classification
SSE - Server-Sent Events
HTML – Hypertext Markup Language
CSS – Cascading Style Sheets
JSON – JavaScript Object Notation
HTTP – Hypertext Transfer Protocol
GCP – Google Cloud Platform
TSV – Tab-Separated Values

Table of Contents

Abstract	i
Abbreviations/Acronyms	iii
Table of Contents	iv
List of Figures	v
1. Introduction	1
2. Literature Review	2
2.1 What is openBIM? Overview of IFC (IfcAlignment) in InfraBIM	2
2.3 Natural Language Processing (NLP & NLU)	9
2.4 Review of Existing IFC Viewers.....	11
3. Methodology	13
3.1 Tools and Technologies used.....	13
3.2 Workflow	14
3.3 IFC Model Preparation.....	15
3.4 Web Application Development.....	19
3.5 Chatbot Setup.....	21
4. Prototype	27
4.1 Detailed Description with Code Snippets of the Web Application	27
4.2 Integrating Dialogflow CX chatbot with the app.....	31
4.2 User Interface Design	39
5. Results	42
5.1 Outcomes of the Implementation	42
5.2 Performance Benchmarks	47
6. Conclusion and Future Development	48
Bibliography	54

List of Figures

Figure 1: buildingSmart	3
Figure 2: Horizontal Alignment. Source: FHWA (WIKI.OSAARCH, 2024).....	5
Figure 3: Vertical Alignment. Source: FHWA (WIKI.OSAARCH, 2024).....	5
Figure 4: IFC 4.3.2 semantic alignment model (WIKI.OSAARCH, 2024).....	6
Figure 5: IFC 4.3.2 geometric representation of horizontal and vertical alignment (WIKI.OSAARCH, 2024).....	7
Figure 6: Horizontal Plan View of Alignment (WIKI.OSAARCH, 2024)	8
Figure 7: Vertical View of the Profile (WIKI.OSAARCH, 2024)	8
Figure 8: 3D View of Alignment (WIKI.OSAARCH, 2024).....	8
Figure 9: Google Dialogflow Agent architecture (Building a "ChatBot" for Scientific Research - Scientific Figure on ResearchGate)	10
Figure 10: Integration Architecture from the Google Dialogflow Documentation (Greyling, 2020).....	11
Figure 11: Integration of web based IFC viewer with Conversational Agent: A workflow using Dialogflow for Natural Language Interaction.....	15
Figure 12:The tunnel model in 3D view in Revit	16
Figure 13: The Alignment of the whole model	17
Figure 14: The Profile View of the Alignment.....	17
Figure 15: The Profile Properties.....	18
Figure 16: IFC export extension in Civil 3D	18
Figure 17: Web App Development Phases.....	19
Figure 18: Chatbot Setup in Dialogflow CX	21
Figure 19: The created Intents in Dialogflow	21
Figure 20: "Isolate" intent addition.....	22
Figure 21: "Isolate" Webhook creation	23
Figure 22: Default Start Flow of the conversation.....	24
Figure 23: Route definition.....	25
Figure 24: Chatbot UI source code	25
Figure 25: Project folder contents.....	27
Figure 26: User Interface of the application	39
Figure 27: Chatbot UI component	40
Figure 28: Properties Panel.....	41
Figure 29: The Tunnel Model loaded in the Application	42
Figure 30: Visualized Alignments in the Application	42
Figure 31: Horizontal Alignment and Cross-Section Panel	43
Figure 32: The Profile View Panel.....	43
Figure 33: Chatbot Agent Start	44
Figure 34: Isolate function executed by Chatbot	45
Figure 35: "Show all hidden elements" function executed by Chatbot.....	45
Figure 36: Chatbot response with the IFC class of the selected element.....	46
Figure 37: Exported Cross Section Panel as DXF file.....	47
Figure 38: Performance Indicator Panels.....	47
Figure 39: GitHub Repository of the Project	51

1. Introduction

Building Information Modeling (BIM) has profoundly transformed the construction industry by introducing a collaborative approach to project management. It allows for the continuous sharing and utilization of data throughout a building's entire lifecycle which starts from initial design through to operation and maintenance. This collaborative process, which occurs between various stakeholders, is driven by the "Information" component within BIM. These data are actively used to improve decision-making by minimizing errors and enhance coordination among teams.

An important component for many BIM implementations in the AECO industry is the Industry Foundation Classes (IFC) standard developed by BuildingSmart. IFC is an open and standardized file format that facilitates the exchange of BIM data across various software applications. This interoperability is crucial in an industry where different stakeholders often rely on different tools in their projects. IFC ensures that information can be shared without compatibility issues, thus supporting better communication and coordination—critical factors for successful project outcomes.

However, there are some notable challenges in IFC visualization. As infrastructure projects become more sophisticated, there is an increasing need for advanced tools to handle and interpret large volumes of data. Many existing IFC viewers do not provide the detailed and synchronized visualizations required for complex infrastructure models. These tools often lack the ability to simultaneously display model alignments, profiles, and cross-sections in a web browser, leading to potential inefficiencies and in project planning and execution.

To address these challenges, this study proposes to develop a web-based application designed to enhance the visualization and interaction of IFC data. The study will create a tool using open-source JavaScript libraries that offers advanced alignment visualization features. This application aims to fill the existing gaps in current IFC viewers, especially those web-based.

In addition, the study will show how to integrate Natural Language Processing (NLP) through a chatbot powered by Dialogflow CX. This feature will enable users to interact with the application using natural language commands and queries and will make the viewer more intuitive and accessible.

The research will also demonstrate the practical application of the tool by describing the complete workflow— starting from exporting the IFC model with alignments, setting up the Conversational Agent system and programming the web-based viewer. This demonstration will highlight the innovative technological tools and their practical benefits.

Furthermore, the study will assess the performance and usability of the application through a simple testing. This evaluation will provide insights into how well the tool meets user needs, its effectiveness in various conditions and the current limitations.

In the conclusion chapter, the thesis interprets the results, examines their implications for the AECO industry, acknowledges the limitations of the study, and suggests potential areas for future research. This study contributes to the advancement of openBIM technologies by providing an interactive and efficient tool for visualizing infrastructure models, ultimately aiming to improve BIM management practices in the AECO industry.

2. Literature Review

2.1 What is openBIM? Overview of IFC (IfcAlignment) in InfraBIM

openBIM extends the benefits of BIM (Building Information Modeling) by improving the accessibility, usability, management and sustainability of digital data in the built asset industry. At its core, openBIM is a collaborative process that is vendor neutral. openBIM processes can be defined as sharable project information that supports seamless collaboration for all project participants. openBIM facilitates interoperability to benefit projects and assets throughout their lifecycle. (buildingSMART, 2024)

By adhering to international standards and working procedures, openBIM® extends the breadth and depth of the use of BIM by creating common alignment and language. Technical applications developed for openBIM® improve the management of data and eliminate disconnected workflows. (buildingSMART, 2024).

The principles of openBIM recognize that:

1. Interoperability is key to the digital transformation in the built asset industry (buildingSMART, 2024): Effective data exchange between different software systems and stakeholders is essential for the successful implementation of BIM. Interoperability ensures that all parties can access and utilize the same data, facilitating better coordination and reducing errors and rework (Eastman, Teicholz, Sacks, & Liston, 2011)
2. Open and neutral standards should be developed to facilitate interoperability (buildingSMART, 2024): Standards such as IFC, developed by buildingSMART, provide a common language for BIM data. These standards are vendor-neutral, ensuring that data can be shared and understood regardless of the software platform used (buildingSMART, 2024).
3. Reliable data exchanges depend on independent quality benchmarks (buildingSMART, 2024): Independent certification and quality assurance processes, such as those provided by buildingSMART's certification programs, help ensure that BIM data meets high standards of accuracy and reliability. This fosters trust and confidence among project stakeholders.
4. Collaboration workflows are enhanced by open and agile data formats (buildingSMART, 2024): Open data formats enable more flexible and efficient collaboration. They allow different tools and applications to work together seamlessly, supporting dynamic and adaptive workflows that can respond to changing project needs (Pauwels, Zhang, & Lee, 2017).
5. Flexibility of choice of technology creates more value for all stakeholders (buildingSMART, 2024): By adopting open standards, stakeholders are not locked into a single vendor's ecosystem. This flexibility allows them to choose the best tools for their specific needs, leading to better outcomes and more innovation in the industry (Eastman, Teicholz, Sacks, & Liston, 2011).
6. Sustainability is safeguarded by long-term interoperable data standards (buildingSMART, 2024): Open standards ensure that data remains accessible and usable over the long term, even as technology evolves. This is critical for the sustainable management of built assets, allowing for ongoing maintenance, renovation, and reuse of data without the risk of obsolescence (Eastman, Teicholz, Sacks, & Liston, 2011).

BIM (Building Information Modeling) for transportation infrastructure has earlier been studied by Costin et al. (2018) (Costin, Adibfar, Hu, & Chen, 2018). The results showed the use of BIM for transportation infrastructure has been increasing with especially focusing on roads, highways, and bridges. Also they reveal a major need for a standard neutral exchange format and schema to promote interoperability. Benefit assessments were also found: the use of BIM instead of traditional documents was estimated to lead to economic and technical benefits. Also it was stated that application of BIM can help general contractors to reduce their risks and diminish the associated costs. (Heikkila, Kolli, & Rauhala, 2022)

Infra BIM is an acronym for Infra Built Environment Information Model, which includes the infrastructure information model and related structures and environment information (K., 2014). According to Costin et al. (Costin, Adibfar, Hu, & Chen, 2018) literature review, many publications have been published since 2002 to 2017 where BIM has studied various infrastructures related to the transport sector, such as bridges, roads, and mass transit. BIM has found to have unlimited potential for improving infrastructure, but there is still challenges interoperation and data exchange. (Kolli & Heikkila, 2020)

The introduction of infrastructure modeling has been promoted in countries by many parties when the benefits of its use have come to the attention of governments. At the European level, the use of infrastructure modeling is a requirement in public sector projects, in addition to Finland, for example in Sweden, Norway, and Great Britain. The Nordic countries have been at the forefront of introducing open infrastructure modeling. (Kempainen, Kolli, & Heikkilä, 2024).

Several InfraBIM software solutions on the market today are specifically designed to support the complex demands of infrastructure projects, among them:

Autodesk Civil 3D (offers robust tools for site design, grading, drainage etc.), Bentley MicroStation (known for its scalability and customization options), Tekla Structures (specialized in structural engineering), Autodesk InRoads (suitable for early-stage design and infrastructure planning), Bentley OpenRoads Designer (a comprehensive solution for road and highway design), Solibri Model Checker (not a design tool, but essential for quality control and clash detection), Trimble Connect, Autodesk Revit. (RootsBIMLLC, 2024)

In general, IFC, or “Industry Foundation Classes”, is a standardized, digital description of the built environment, including buildings and civil infrastructure. It is an open, international standard ([ISO 16739-1:2018](#)), meant to be vendor-neutral, or agnostic, and usable across a wide range of hardware devices, software platforms, and interfaces for many different use cases. The IFC schema specification is the primary technical deliverable of buildingSMART International to fulfill its goal to promote openBIM®. (buildingSMART, 2024)



Figure 1: buildingSmart

More specifically, the IFC schema is a standardized data model that codifies, in a logical way:

...the identity and semantics (*name, machine-readable unique identifier, object type or function*)

...the characteristics or attributes (*such as material, color, and thermal properties*)

...and relationships (including locations, connections, and ownership)

...of objects (like columns or slabs)

...abstract concepts (performance, costing)

...processes (installation, operations)

...and people (owners, designers, contractors, suppliers, etc.).

The schema specification can describe how a facility or installation is used, how it is constructed, and how it is operated. IFC can define physical components of buildings, manufactured products, mechanical/electrical systems, as well as more abstract structural analysis models, energy analysis models, cost breakdowns, work schedules, and much more. (buildingSMART, 2024)

This thesis' case study IFC model has the schema IFC 4.3.2, which is an official version approved by ISO 16739-1: 2024. This schema version is a significant update which reflects the evolving needs in BIM and openBIM communities. A particular key feature is the support for Infrastructure projects, including:

- Better accommodating a wide range of infrastructure elements, such as roads, railways, bridges, tunnels, and utilities.
- Improved handling of geospatial information helps in accurately representing the location and context of infrastructure elements, which is crucial for large-scale projects and site analysis.
- Enhanced support for alignments.

In IFC, an alignment is a geometric representation that describes the trajectory or path of linear infrastructure elements. It serves as the fundamental reference for designing and positioning these elements within a project. Alignments are critical for defining the layout of roads, railways, runways, pipelines, and other linear infrastructure components.

There are 2 types of alignments in IFC: horizontal and vertical.

A horizontal alignment typically consists of straight lines called tangents that are connect by curves. The curves are usually circular arcs. Easement or transition curves in the form of spirals can be used to provide gradual transitions between tangents and circular curves. The horizontal alignment is a plan view curvilinear path in a Cartesian coordinate system aligned with East and North. Positions along an alignment, measured along the plan view projection of the 3D curve alignment curve, are denoted with stations or chainage. The example alignment has four tangent runs connected by three horizontal curves. The alignment begins at Station 100+00. The first curve begins at Station 119+56.79 which is 19 chains plus 56.79 links from the origin. (WIKI.OSAARCH, 2024)

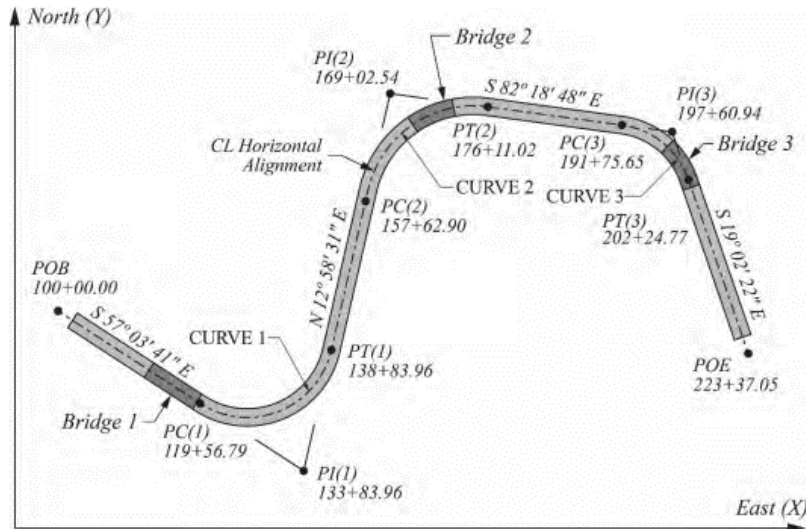


Figure 2: Horizontal Alignment. Source: FHWA (WIKI.OSAARCH, 2024)

A vertical alignment consists of straight sections of grade lines connected by vertical curves. The vertical curves are typically parabolas, though sometimes circular arcs or clothoid curves are used. A vertical alignment is defined along the curvilinear path of a horizontal alignment in a *Distance Along, Elevation* coordinate system. Combined, the horizontal and vertical alignments define a 3D curve. This combination of two 2D curves is sometimes referred to as a 2.5D geometry. The example vertical alignment consists of five gradients connected by four parabolic vertical curves. (WIKI.OSAARCH, 2024)

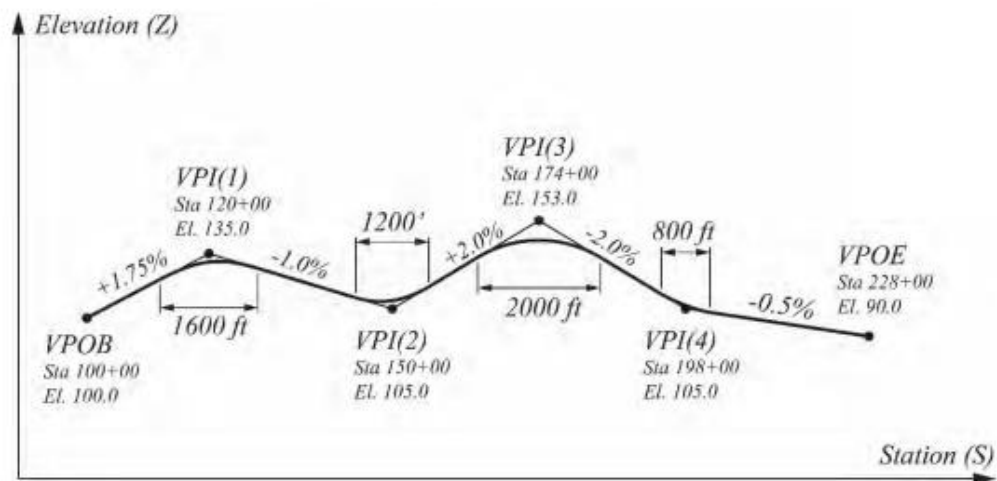


Figure 3: Vertical Alignment. Source: FHWA (WIKI.OSAARCH, 2024)

The semantic definition of alignment allows for the alignment to be described as close as possible to the terminology and concepts used in a business context. Examples include horizontal, vertical and can't

layouts, stationing, and anchor points for domain specific properties such as design speed, cant deficiency, superelevation transitions, and widenings. (WIKI.OSAARCH, 2024)

Specialized applications can analyze alignments using semantic information. Alignments can be evaluated against design criteria such as speed requirements, sight distances, and maximum gradient, to name a few.

The IfcAlignment semantic definition is composed of an instance of IfcAlignmentHorizontal and IfcAlignmentVertical through an IfcRelNests relationship.

The IfcAlignmentHorizontal and IfcAlignmentVertical are in turn composed of one or more IfcAlignmentSegment entities through IfcRelNests relationships.

The IfcAlignmentSegment models the segment design parameters in a subtype of IfcAlignmentParameterSegment; IfcAlignmentHorizontal and IfcAlignmentVertical in this case. The semantic definition model is shown in Figure 4. (WIKI.OSAARCH, 2024)

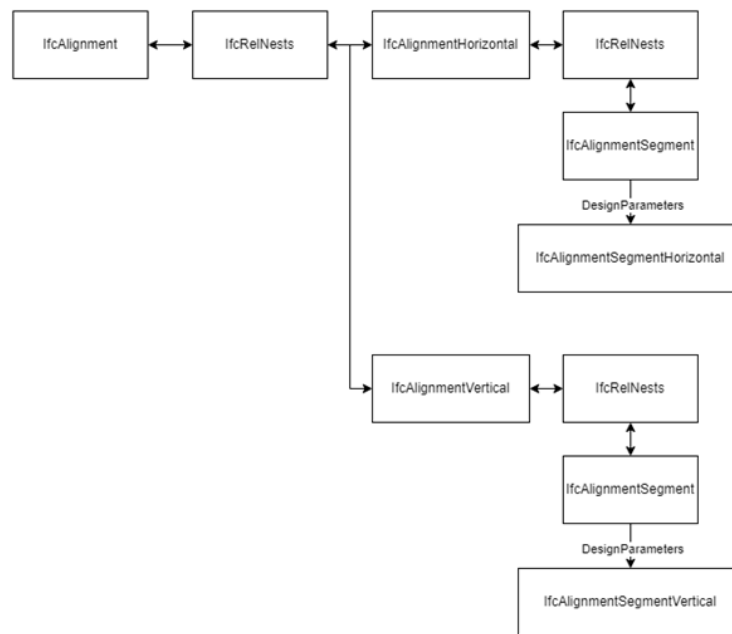


Figure 4: IFC 4.3.2 semantic alignment model (WIKI.OSAARCH, 2024)

The semantic definition of an alignment describes its business attributes (e.g. design speed) while the geometric definition describes the precise geometry (e.g. line from point A to point B followed by an arc of radius R starting at point B and ending at point C).

The geometric representation of the IfcAlignment consists of an IfcShapeRepresentation for the plan view horizontal alignment as well as a 3D curve for the combined horizontal and vertical alignment. This is illustrated in Figure 5.

Geometrically, the horizontal alignment is defined by an IfcCompositeCurve and the vertical alignment is defined by an IfcGradientCurve. The horizontal alignment is the basis curve for the vertical alignment. (WIKI.OSAARCH, 2024)

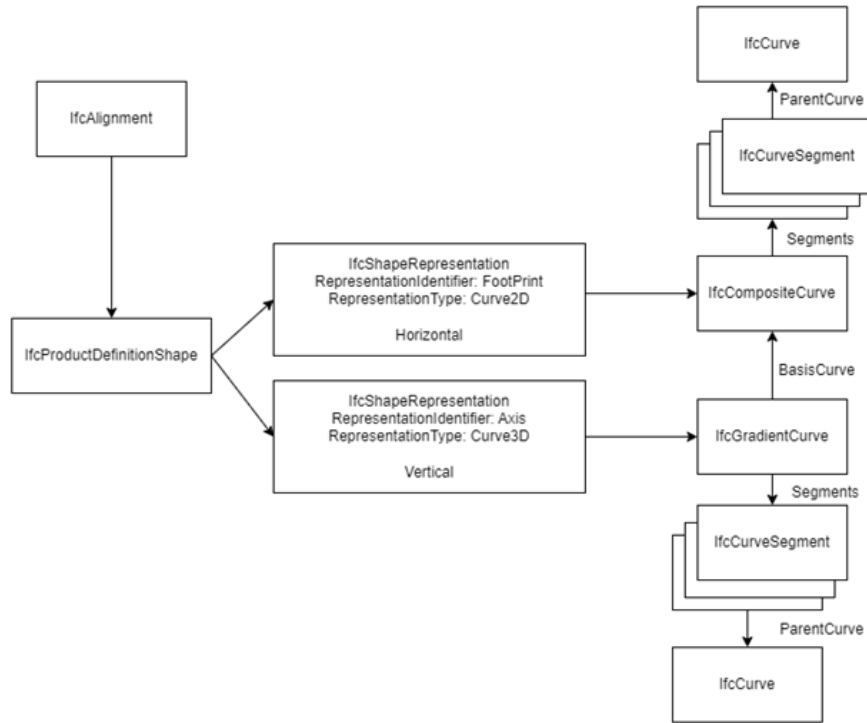


Figure 5: IFC 4.3.2 geometric representation of horizontal and vertical alignment (WIKI.OSAARCH, 2024)

The geometric elements are modeled with IfcCurveSegment. IfcCurveSegment cuts a segment from a parent curve and places it in the alignment coordinate system. A horizontal circular arc is modeled with an IfcCurveSegment that cuts an arc from an IfcCircle. The line and curve segments are cut from their IfcLine and IfcCircle parent curves, respectively. The process of defining a parent curve, cutting a segment from that curve, and placing the cut segment into the alignment is repeated to define the entire horizontal and vertical alignments.

All the horizontal alignment IfcCurveSegment objects are then combined to create an IfcCompositeCurve to complete the plan view geometric representation of the horizontal alignment.

A similar process is used to create vertical alignment. Parent curves are defined and IfcCurveSegment segments are defined that cut a portion of the parent curve and position it in the XY plane where X is distance along the horizontal alignment and Y is elevation. The vertical alignment IfcCurveSegment objects are then used to create an IfcGradientCurve.

The IfcGradientCurve uses the horizontal alignment IfcCompositeCurve as its basis curve, which means the horizontal coordinates of the vertical alignment are taken from the horizontal alignment curve as illustrated in Figures 2 and 3.

The images below (Figure 6,7 and 8) show the three different ways an alignment can be visualized. We can visualize the horizontal and vertical alignment in a 2D grid with scales, and also the 3D geometry in the same way (WIKI.OSAARCH, 2024).

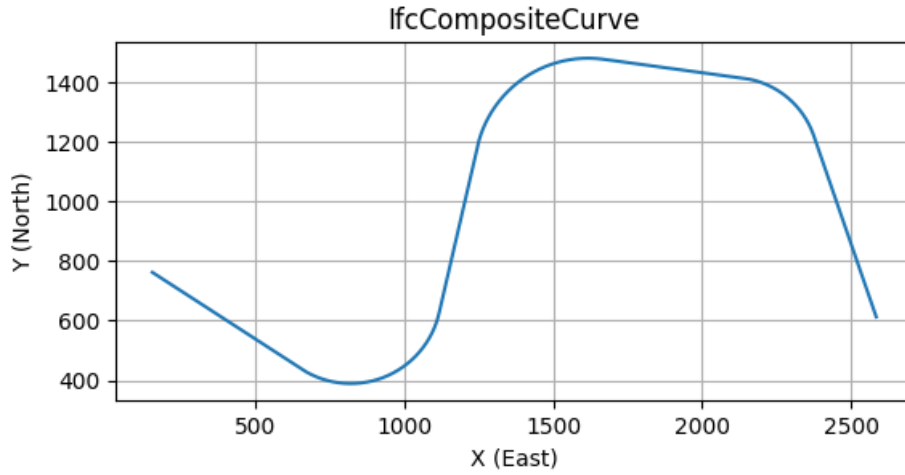


Figure 6: Horizontal Plan View of Alignment (WIKI.OSAARCH, 2024)

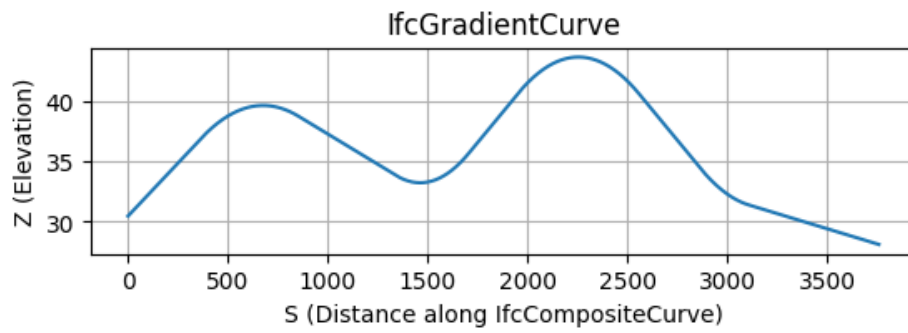


Figure 7: Vertical View of the Profile (WIKI.OSAARCH, 2024)

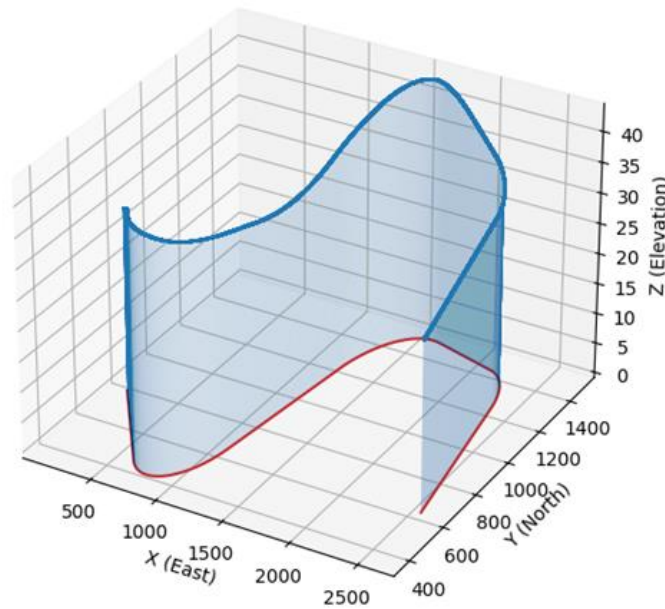


Figure 8: 3D View of Alignment (WIKI.OSAARCH, 2024)

2.3 Natural Language Processing (NLP & NLU)

Artificial intelligence deals with the science of developing intelligent models and machines that can mimic human intelligence processes. Its application has been transforming diverse fields and improving productivity. The architecture engineering and construction (AEC) industry is not left out in applying AI to solve problems or improve conventional approaches. There has been a surge in the application of AI in the AEC industry over the last decades (Abioye, et al., 2021). Consequently, studies have reviewed extant literature on AI in the industry (Wu & Dai, 2021), (Darko, et al., 2020), (Akinosho, et al., 2020), (Bilal, et al., 2016). A fast and emerging new research area in AI is Conversational Artificial Intelligence (Conversational AI) which is aimed at improving human–computer interaction. (Saka, et al., 2023) The dream of developing machines that can interact with man permeates time to the early ancient Greek creation of robots and the imagination of artificial life that is ‘made, not born (Mayor, 2018). Conversational AI is brought about by advancements in Natural Language Processing (NLP) which has been revolutionizing the way humans relate with computers. (Saka, et al., 2023).

Natural language understanding (NLU) is a complex subdomain of natural language processing (NLP) that deals with the task of understanding human language. It involves intent classification and entity extraction from input (Liu, Eshghi, Swietojanski, & Rieser, 2021). Intent classification (IC) is the process of identifying the user’s sentiment and determining the user’s objective. Traditional IC models employed approaches such as Hidden Markov Models (HMM), Decision Trees (DT) and others. With the advancement in machine learning, deep learning is now employed for intent classifications. Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), bidirectional Long Short-Term Memory (biLSTM), and shallow feedforward networks are used to improve intent classification (McTear, 2021).

Entity extraction or named entity recognition (NER) deals with extracting entities and classifying them into predefined classes. Conditional Random Fields (CRFs) were commonly employed for NER, and new approaches such as CNNs, and biLSTM improves entity extraction. The NLU component is evaluated by comparing output from the component with a reference representation. Metrics such as sentence accuracy, concept accuracy, confusion matrix, precision, recall and F1 (McTear, 2021).

Conversational platforms include two main families:

- Extension of a product: These need an existing product (software and/or hardware) to work, e.g., Actions on Google or Skills for Amazon Echo.
- Standalone services: These provide a series of facilities to create a wide range of conversational interfaces on one platform, typically integrated in "suites" of cloud services, e.g., Dialogflow, IBM Watson, wit.ai. (Russis & Roffarello, 2023)

Dialogflow, originally known as api.ai, is a California-based startup founded in 2010 and acquired by Google in 2016. It offers a platform to create natural and engaging conversational experiences. Users can integrate it easily with popular services like Telegram, Facebook Messenger, and Google Assistant. The tool supports multiple languages, including English, Dutch, and Chinese, and provides a REST API along with official SDKs for various programming languages like Java, Python, and Node.js. It’s free for basic usage, making it accessible for a range of applications.

In Dialogflow, each application, referred to as an agent, operates with unique entities and intents.

- Intent: This is the core of how Dialogflow maps user inputs to specific actions. It involves:

- User Input: What the user says or types.
- Action: The operation or task that the agent should perform based on the user's input.
- Response: The reply or information that the agent provides to the user.

Dialogflow offers various pre-built intents that can be enabled and customized to fit specific needs, streamlining the development of conversational interactions.

- Entities:
 - Represent concepts
 - Serve for extracting parameter values from natural language inputs.
 - Should be created only for concepts that require actionable data (Russis & Roffarello, 2023).

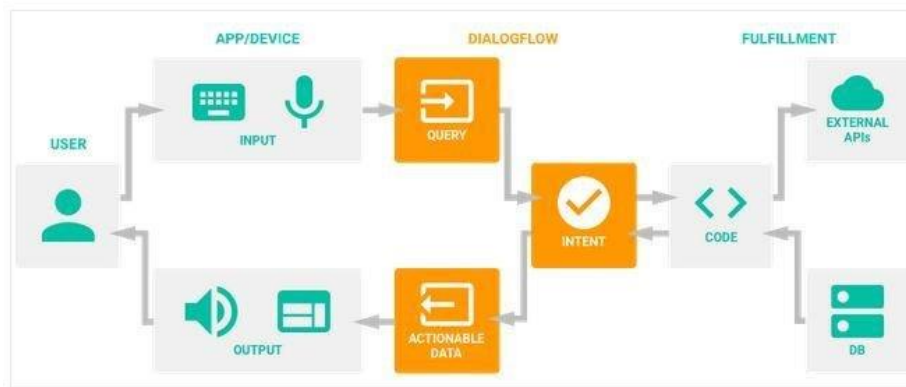


Figure 9: Google Dialogflow Agent architecture (Building a "ChatBot" for Scientific Research - Scientific Figure on ResearchGate)

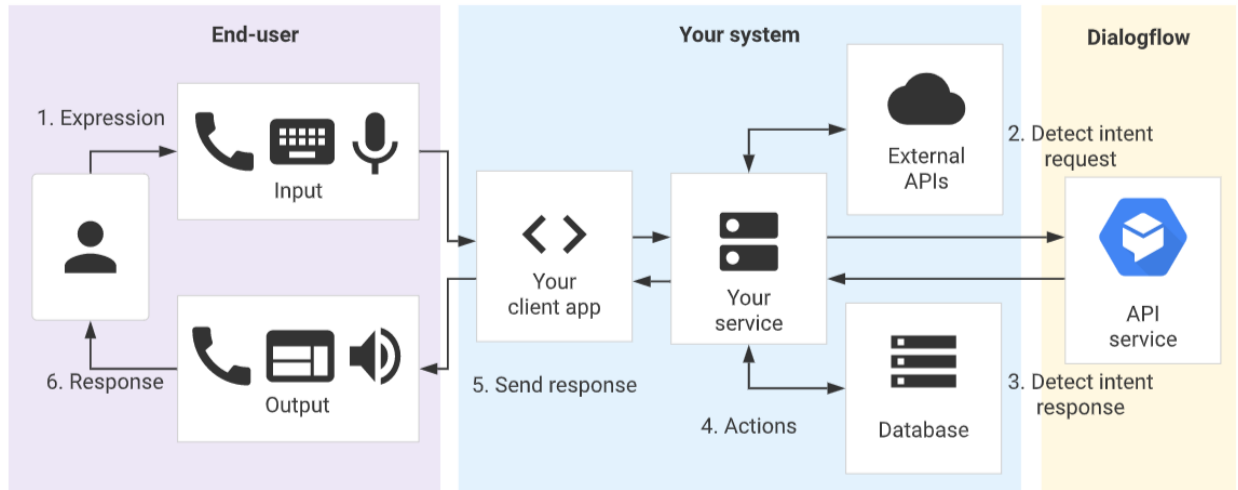


Figure 10: Integration Architecture from the Google Dialogflow Documentation (Greyling, 2020)

2.4 Review of Existing IFC Viewers

In the realm of BIM, a diverse array of IFC viewers is available, each offering varying levels of functionality and accessibility. These viewers are critical tools for visualizing, analyzing, and managing BIM data across different stages of a project. They can be broadly categorized into web-based and desktop-based solutions, with some available for free and others requiring a subscription or one-time purchase. Almost all the free viewers offer basic key features such as model viewing, measurements, clash detections, and some other commercial BIM software provide high level visualization, editing and collaboration. The table below shows a brief description of some of the most used software that support IFC viewing and editing with their key features:

Viewer Name	Web-Based	Desktop-Based	Free	Paid	Key Features
BIM Vision	No	Yes	Yes	No	Model viewing, measurements, plugins
Solibri Anywhere	No	Yes	Yes	No	Model viewing, coordination
Dalux Viewer	Yes	Yes	Yes	Yes	3D model viewing, mobile access
BIMx (Graphisoft)	Yes	Yes	Yes	Yes	Interactive 3D navigation, docs
Tekla BIMsight	Yes	Yes	Yes	No	Model viewing, clash detection
DDS-CAD Viewer	No	Yes	Yes	No	Simple model viewing
usBIM.viewer+	Yes	Yes	Yes	Yes	Clash detection, code checking
xBIM Flex Viewer	Yes	No	Yes	No	Open-source, BIM model analysis
BIMcollab Zoom	Yes	Yes	Yes	Yes	Issue management, clash detection
Revit	No	Yes	No	Yes	Comprehensive BIM software
Navisworks	No	Yes	No	Yes	Model coordination, clash detection
SimpleBIM	No	Yes	No	Yes	Model editing, validation
Trimble Connect	Yes	Yes	Yes	Yes	Collaboration, model viewing

Revizto	Yes	Yes	No	Yes	Real-time issue tracking, coordination
Open IFC Viewer	Yes	No	Yes	No	Open-source, model viewing, measurement, basic editing
Autodesk Viewer	Yes	No	Yes	Yes	Web-based viewing, collaboration
Bimsync	Yes	No	Yes	Yes	Collaboration, model viewing, issue tracking
Speckle	Yes	Yes	Yes	No	Open-source, data exchange, model viewing
BlenderBIM	No	Yes	Yes	No	Open-source, model viewing, basic editing
ACCABIM	Yes	No	Yes	Yes	Model viewing, collaboration, issue tracking

Comparison of BIM Viewers: Web-Based, Desktop-Based, and Key Features

xBIM Flex Viewer, Open IFC Viewer, Speckle and BlenderBIM are open-source viewers, meaning that the software is developed and released with its source code available to the public. These tools are valuable for various flexible and cost-effective solutions and are customized to fit specific needs in the BIM domain, as will be shown in the implementation section of this thesis.

The key benefits of open-source project can be summarized as follows:

- Customization and Flexibility
- Cost-Effectiveness
- Transparency and Security
- Community-Driven Development
- Integration Capabilities
- Educational Value
- Long-Term Viability

The BlenderBIM Add-on exemplifies the power of open-source solutions in the BIM domain. It is an open-source tool that extends the functionality of Blender, a popular open-source 3D modeling and animation software, to support BIM workflows. Blender BIM is built over IfcOpenShell, a library dedicated to parsing and processing IFC data, making it possible to import, export, and manage complex BIM models seamlessly using C++ and Python API. Python and C++ make it possible to explore the limits of IFC parsing and authoring in depth and allow to create customizable scripts based on specific workflows.

Another good example of a successful and new open-source project in the AECO software industry is That Open (formerly known as Ifc.js). It is a powerful JavaScript library on a Github Repository that enables to parse and visualize IFC files within web browser. Another package is being developed by this project to enable IFC editing directly on the browser.

The main drawback when dealing with open-source tools in BIM is the integration challenges, requiring significant technical expertise, which most of the Engineers or Architects working in industry lack. However, the fast development of AI has enabled professionals without coding experience to significantly reduce the learning curve to build tools based on programming language.

3. Methodology

3.1 Tools and Technologies used

This chapter explains all the tools and technologies used to achieve the final IFC viewer. Those include:

- An Autodesk Revit tunnel model provided by a private company
- Autodesk Civil 3D for Alignment creation
- IFC 4.3.2 version
- Frontend Libraries and Technologies
- Backend Libraries and Technologies

Frontend Libraries and Technologies

That Open consists of these packages:

thatopen/components-front - Frontend components for BIM applications.

thatopen/ui - UI components library for BIM applications.

thatopen/ui-obc - UI components related to OBC (Open BIM Components).

web-ifc - a javascript library to read and write ifc files, at native speeds.
Three.js - A JavaScript 3D library used to create and display animated 3D graphics in a web browser.

Stats.js - A JavaScript performance monitor used with Three.js to display the frame rate and other stats.

Backend Libraries and Technologies:

Node.js and Express

Node.js: The server-side JavaScript runtime environment that executes the code.

Express: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

express module: Used to create the server and define routes.

express.static middleware: Serves static files.

HTML and CSS:

Programming language used for creating and styling the user interface.

Web Components - Used to create reusable UI components.

JSON:

JavaScript Object Notation, a lightweight data interchange format.

Used for sending and receiving data between the client and server, especially in POST requests and responses.

HTTP Methods:

GET: Used to serve static files and HTML content.

POST: Used to handle incoming requests with data payloads from clients.

Server-Sent Events (SSE):

A server push technology enabling a server to push real-time updates to the client.

- Custom implementation: Setting headers (Content-Type, Cache-Control, Connection) and handling SSE connections.

Ngrok

Ngrok is a tool that creates secure tunnels to locally hosted web applications, allowing you to expose a local server to the internet. It is widely used for testing webhooks, APIs, and other web services during development without the need to deploy your application to a public server.

Dialogflow:

A natural language understanding platform used to design and integrate a conversational user interface into a mobile app, web application, device, bot, etc.

Webhook endpoints (/ClassWebhook) to handle requests from Dialogflow and send responses.

Integrated Development Environment (IDE): Visual Studio Code

A software application that provides comprehensive facilities to computer programmers for software development, including a source code editor, build automation tools, and a debugger.

3.2 Workflow

The schema below describes the workflow of the IFC web viewer integrated with a conversational agent (chatbot) that utilizes Google Dialogflow for natural language processing and interaction.

System Architecture for IFC App and Dialogflow Integration

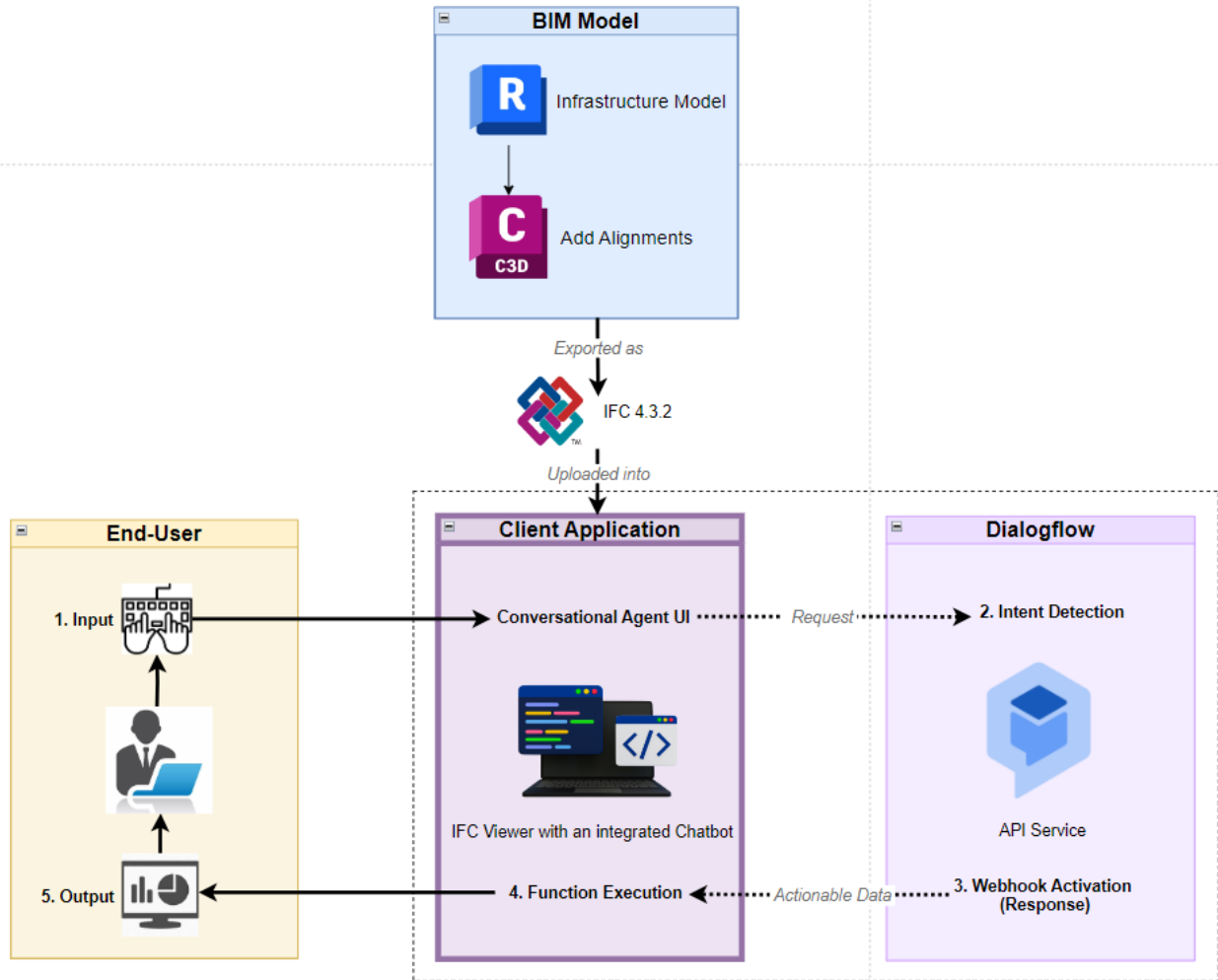


Figure 11: Integration of web based IFC viewer with Conversational Agent: A workflow using Dialogflow for Natural Language Interaction

The “BIM Model” cluster includes the steps of exporting the final IFC infrastructure model file from a Revit Model after Alignments were added in Civil 3D.

The “End-User” cluster represents the interaction between the end-user and the system, which includes the input through the conversational agent and the response from the system.

The “Client-Application” cluster shows the base application for loading the IFC file from the first cluster and the conversational agent UI which makes the interaction between the user and app.

The “Dialogflow” cluster focuses on the natural language processing part of the system. It shows how its API handles the input from the end-user, activates the matching intent and triggers responses via webhooks.

3.3 IFC Model Preparation

The case study used for this section is a tunnel model provided by a construction private company. Autodesk Revit does not support alignments, so the way to do it is to export the model into another software. Autodesk

Civil 3D is a powerful software for civil infrastructure design and documentation. It supports alignment and profile creation, which makes it suitable for this research.

The first step was to export the model as CAD format file. It can be noted that Civil 3D can also import IFC files, so another solution can be to export the Revit model as IFC file and open it there. However, the first solution was chosen in order to avoid any random error, as DWG files are more compact and less likely to make any mistake during export. The export was done from the 3D view in Revit.

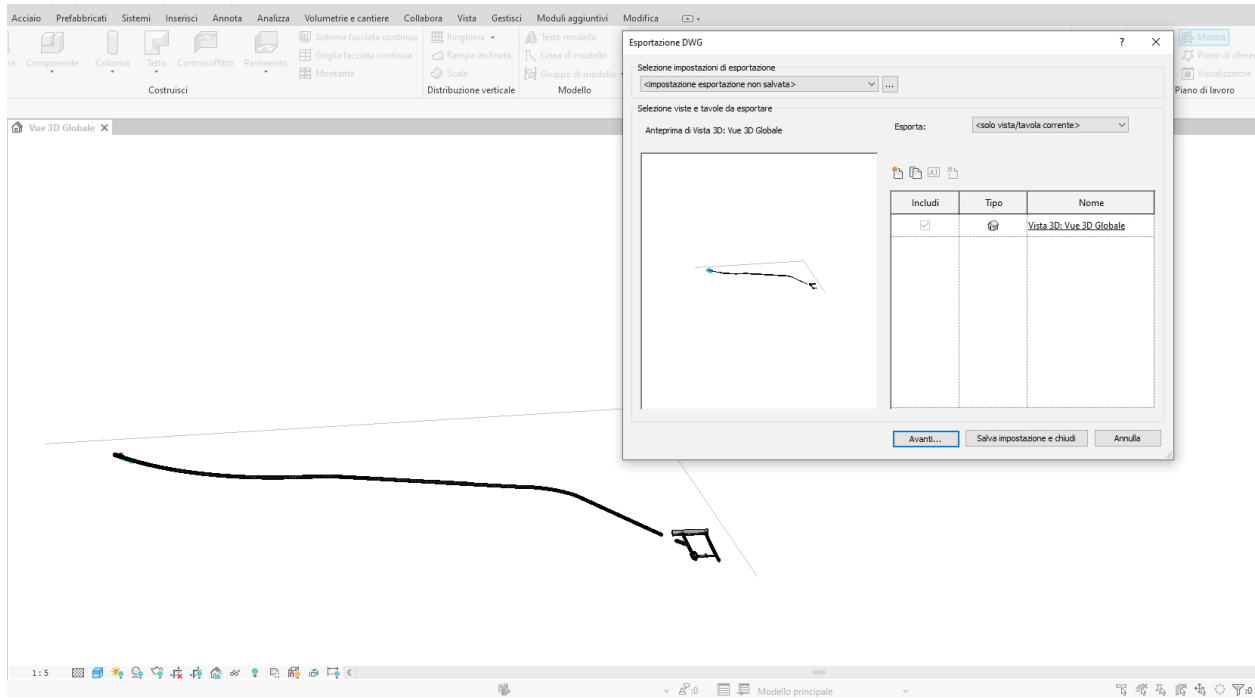


Figure 12: The tunnel model in 3D view in Revit

In Civil 3D every 3D object is opened as an Autocad block, meaning it has all the geometry and properties information exported from Revit. The alignment creation is done as follows:

In the Home tab, Create Design section, the Alignment Creation tool is located. It allows you to draw lines, curves and spirals. There are many other ways to create alignments, such as to create it from an existing line, Autocad Block, points etc. Since a precise feature line that shows the alignment position is missing in this model, a free draw by hand was used to create the alignment based on the geometry of the model. It should be noted that this is not the best way to have accurate information about the alignment, however the purpose of the thesis is to provide the tools to visualize the alignment and profile in a web based IFC viewer. Their accuracy can be improved or properly exported in the IFC if the official CAD drawings of the model were provided.

Also everything else like design speed, direction etc. was kept as a default value created by the software.

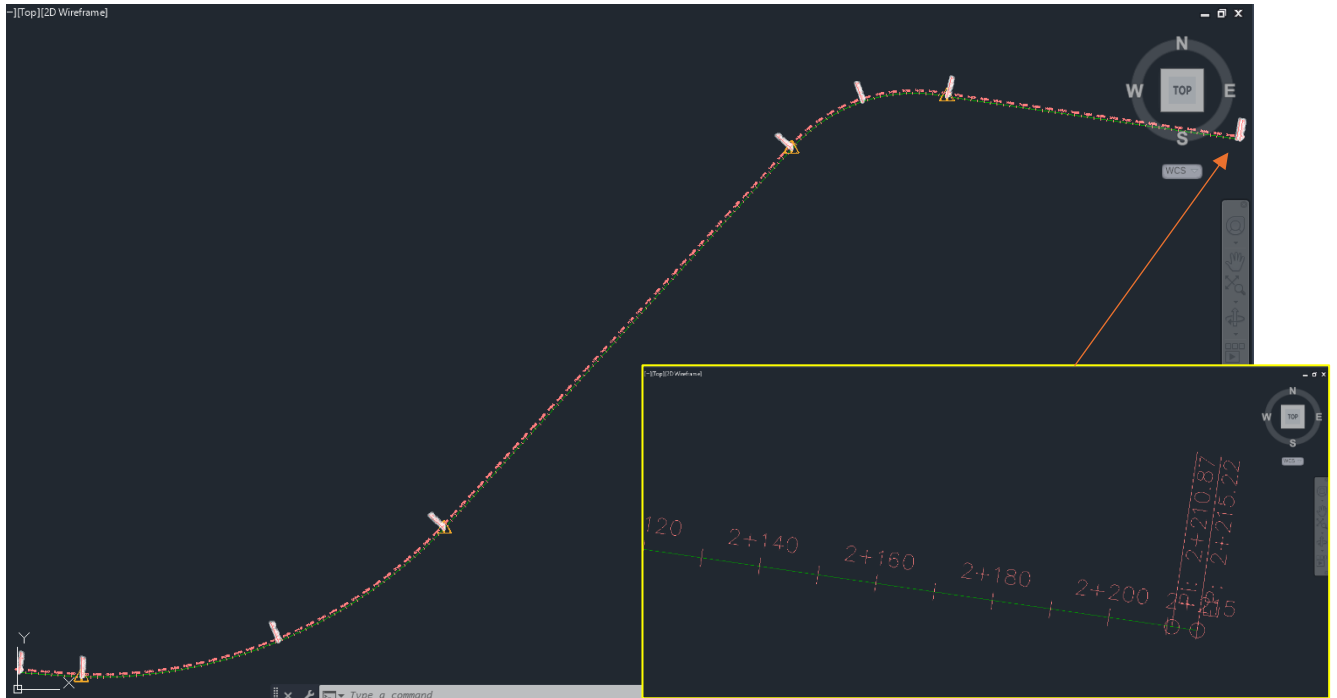


Figure 13: The Alignment of the whole model

The alignment starts at 0+000.00 and ends at 2+215.22m, meaning it has a length of 2215.223m. It is composed of 3 lines and 2 clothoids (curves).

The next step is to create the Profile of our model. Below the alignment creation tool is the Profile one. The methodology to create the profile was similar to the alignment. There are many options to create it from the existing objects, like using AutoCAD blocks that we have in our model. The software creates the best fit profile based on the geometry of those blocks. The result is shown in figure 14:

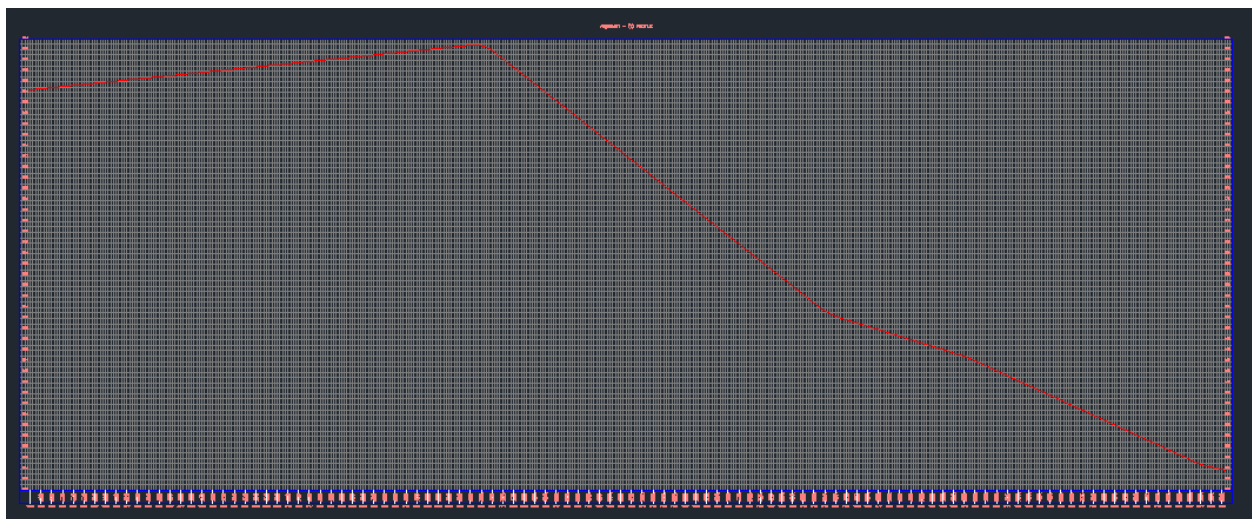


Figure 14: The Profile View of the Alignment

Name	Description	Type	Data Source	Offset	Update Mode	Layer	Style	Station Start	Station End	Elevation Minimum	Elevation Maximum	Alignment
Profile - (1)				0.000m			Existing Grou...	0+000.00m	2+225.00m	623.745m	702.862m	Alignment - (1)

Figure 15: The Profile Properties

The profile is made of several components. In this case they were chosen to export as parabolas. Now in the model we have both a horizontal alignment and the profile of its elevation.

The export to IFC 4.3.2 was done using the IFC exporter extension of Civil 3D (Figure 16). The software has a build-in IFC exporter which supports 4x1 and 2x3 IFC schema, so for this case the installing of the Add-in was necessary.

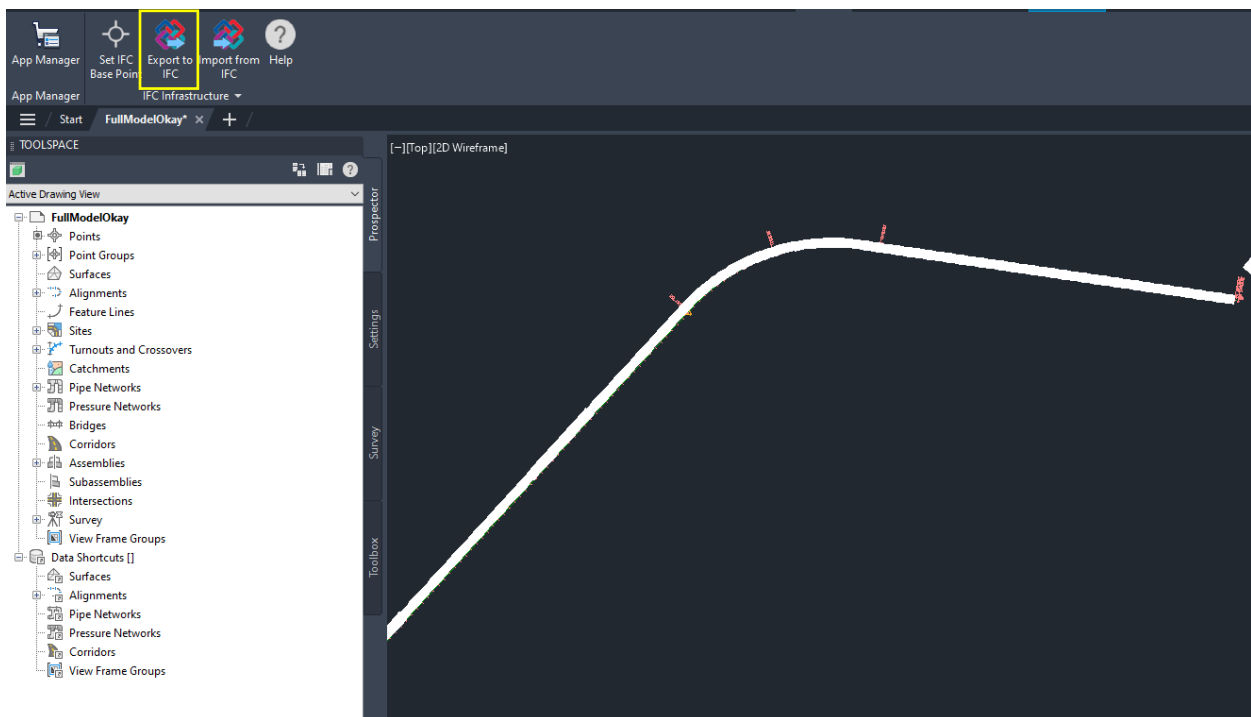


Figure 16: IFC export extension in Civil 3D

3.4 Web Application Development

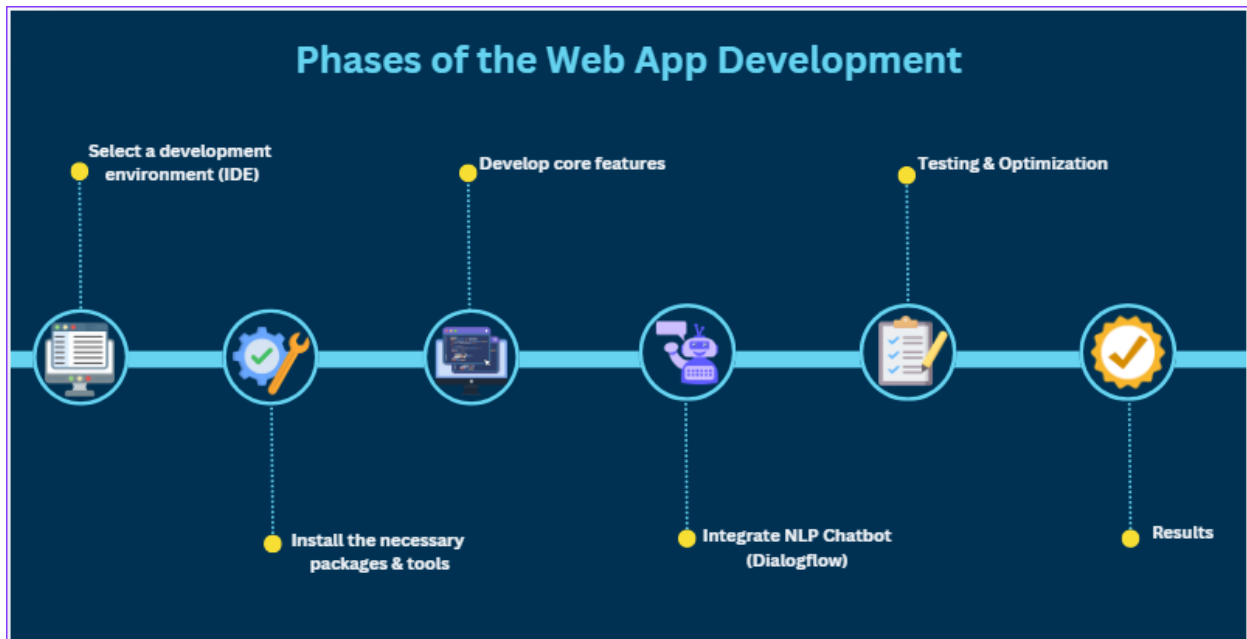


Figure 17: Web App Development Phases

The first things that were imported in the app were the necessary libraries to build the visualization features. In order to use the code for their development, ThatOpen packages and their dependencies were installed by using Node.js commands such as `npm install` and `npm run dev`. All the dependencies of the application should be defined inside a file called `package.json`.

The easiest way to use That Open libraries and their dependencies is to run in the terminal of the opened folder:

`npm create bim-app@latest`

This creates a template with all the files necessary to run an app in the browser. The `package.json` file with all the dependencies to be installed via “`npm install`” are as follows:

```
{
  "name": "bim-app",
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "preview": "vite preview"
  },
  "devDependencies": {
    "@types/three": "0.160.0",
    "@typescript-eslint/eslint-plugin": "7.2.0",
    "@typescript-eslint/parser": "7.2.0",
    "eslint": "8.57.0",
    "eslint-config-airbnb-base": "15.0.0",
```

```

"eslint-config-prettier": "9.1.0",
"eslint-plugin-import": "2.29.1",
"eslint-plugin-prettier": "5.1.3",
"typescript": "5.2.2",
"vite": "5.2.0",
"webpack-cli": "^5.1.4"
},
"dependencies": {
"@thatopen/components": "~2.1.0",
"@thatopen/components-front": "~2.1.0",
"@thatopen/fragments": "~2.1.0",
"@thatopen/ui": "~2.1.0",
"@thatopen/ui-obc": "~2.1.0",
"dxf-writer": "^1.18.4",
"express": "^4.19.2",
"jszip": "3.10.1",
"three": "0.160.1",
"ts-loader": "^9.5.1",
"web-ifc": "^0.0.56",
"webpack": "^5.93.0"
}
}

```

The command “npm run dev” is used to run a custom script defined in the “scripts” section of the package.json file of a Node.js project. It is used to start a development server or run development-specific tasks.

In this project, “npm run dev” starts the Vite development server. Vite is a modern front-end build tool that offers a faster and leaner development experience for modern web projects. The command “tsc” is the TypeScript compiler command, which compiles TypeScript files into JavaScript. Since the code is written in TypeScript, it needs to be bundled into optimized static assets that can be deployed. This is done by the command “vite build”.

“vite preview” starts a local static server to serve the built files, allowing you to preview the production build of your project. It helps in verifying that the build works correctly before deploying.

Since all the basic tools are installed for the application, before describing in details how the alignment visualization was created, a description of the Dialogflow CX setup will be provided in this section. Then everything will be covered in detail in the implementation chapter.

3.5 Chatbot Setup

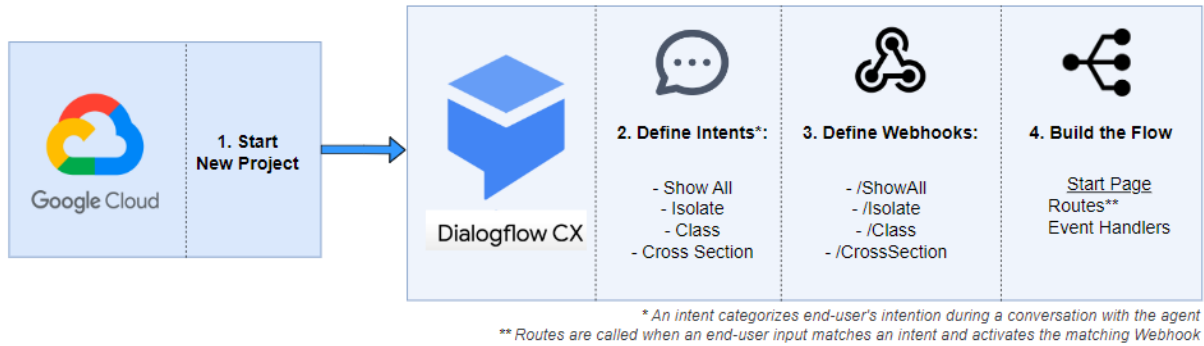


Figure 18: Chatbot Setup in Dialogflow CX

Google Cloud Platform (GCP) is a suite of cloud computing services offered by Google. It runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail, Google Drive, and YouTube. GCP provides a range of services for computing, storage, networking, big data, machine learning, and more. Dialogflow CX (Customer Experience) is a product within GCP that provides a suite of tools for building conversational interfaces. It is part of Google's AI and machine learning offerings and is designed to create advanced conversational experiences for chatbots, virtual agents, and interactive voice response (IVR) systems. Firstly, the intents for this project were defined as shown in Figure 19:

Display name	Labels	# of Training phrases	Last modified	
<input type="checkbox"/> Default Welcome Intent		17	Apr 20, 2024 04:28 PM	
<input type="checkbox"/> Default Negative Intent		0	Apr 20, 2024 04:28 PM	
<input type="checkbox"/> ListPropertySets		3	Apr 21, 2024 06:12 PM	
<input type="checkbox"/> Show All		2	Jul 3, 2024 10:58 PM	
<input type="checkbox"/> Class		3	Jul 20, 2024 04:31 PM	
<input type="checkbox"/> Isolate		2	Jul 3, 2024 10:52 PM	
<input type="checkbox"/> CrossSection		4	Jun 11, 2024 03:34 PM	

Figure 19: The created Intents in Dialogflow

The intents added to the Default Intents are: Show All, Class, Isolate, CrossSection. It will be provided in the next chapters what are these Intents used for. To see how an Intent is created, the “Isolate” intent will be described as an example in Figure:

← Intent
Save
Cancel

An intent categorizes end-user's intention during a conversation with the agent (for example, schedule an appointment). [Learn more](#)

Display name *
 Isolate

Labels ?

Description
 Describe the purpose of the intent. For example: This intent is triggered when user asks a payment question.

Training phrases

When a user says something similar to a training phrase, Dialogflow matches it to the intent. You don't have to create an exhaustive list. Dialogflow will fill out the list with similar expressions. To extract parameter values, use [annotations](#) with available [system](#) or [custom](#) entity types.

Type a training phrase and press 'Enter' Add

Skip auto annotation ?
↑
🗑

🔍 Search Search training phrases

<input type="checkbox"/> Training phrases	# words	
<input type="checkbox"/> isolate the selected element	4	🗑
<input type="checkbox"/> Isolate	1	🗑

Figure 20: "Isolate" intent addition

It starts by naming the Intent in the Display name field. Then training phrases are provided. The powerful feature of Dialogflow is that it uses NPL to match another similar phrase to the intent, so it is not necessary to provide an exhaustive list. In this case two training phrases are created: "Isolate the selected element" and "Isolate". However, even if the user says something similar to those, for example "Can you isolate this element", the system will match to the defined Intent for those training phrases. The more phrases provided the better will the system match the intent.

The same steps were used to create the other intents.

Since this project deals with HTTP callbacks, webhooks are needed to create the communication between Dialogflow and the server which hosts the web application.

Webhook Save Cancel

A webhook is a web server endpoint that you create and host. [Learn more](#)

Enabled

Display name*
Isolate

Webhook timeout
5
Enter a number (in seconds)

Type
Generic web service

Webhook URL*
https://3c87-151-60-60-67.ngrok-free.app/Isolate
Enter a publicly available HTTPS URL to your webhook

Subtype
Standard

Figure 21: "Isolate" Webhook creation

As shown in Figure x, the "Isolate" webhook will be used to be linked with the same Intent when the latter is activated. There are a lot of advanced features provided in the Webhook section which are not used for this project. Among them are Authentication methods for secure connections, CA certificates for HTTPS verification, Third party authorization etc.

A very important field is the Webhook URL. If the URL of the application is wrong, the POST requests made to the server will not happen. A free Ngrok link was used in this case, which exposes a local URL to a Public one, thus providing the ability to make POST requests from other devices on web.

After the creation of Intents and Webhooks, the next step is to create the start flow as shown in Figure 22. This will determine how the Agent will respond to user input texts. Inside the Start Page are Routers, which define the different possible intents that can be triggered and Event handlers which handle unexpected or error conditions during the conversation.

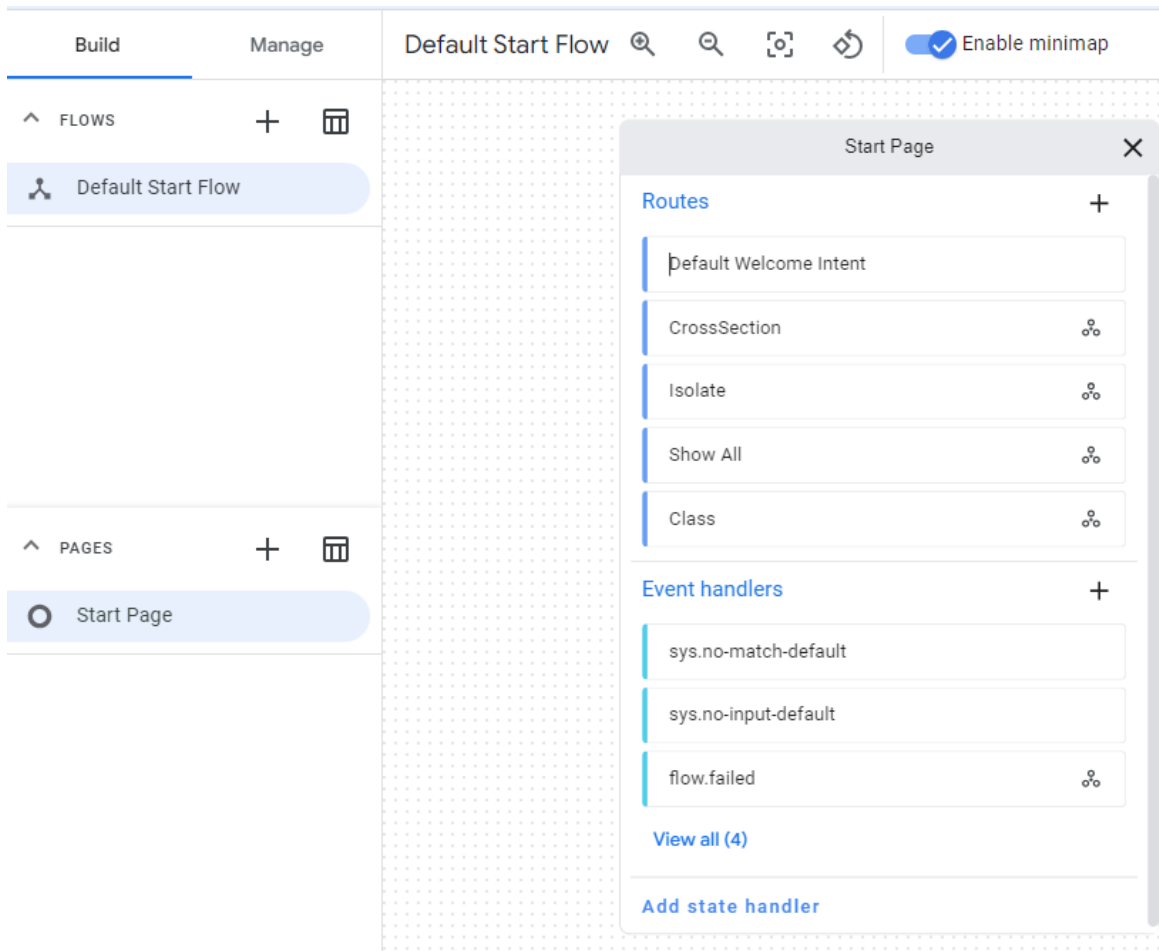


Figure 22: Default Start Flow of the conversation

Each route must be configured to enable the Webhook, which should match the respective intent. For example, for the “Isolate” intent we want to enable the “Isolate” Webhook, in order to make the proper POST request in the server that will activate the isolate function.

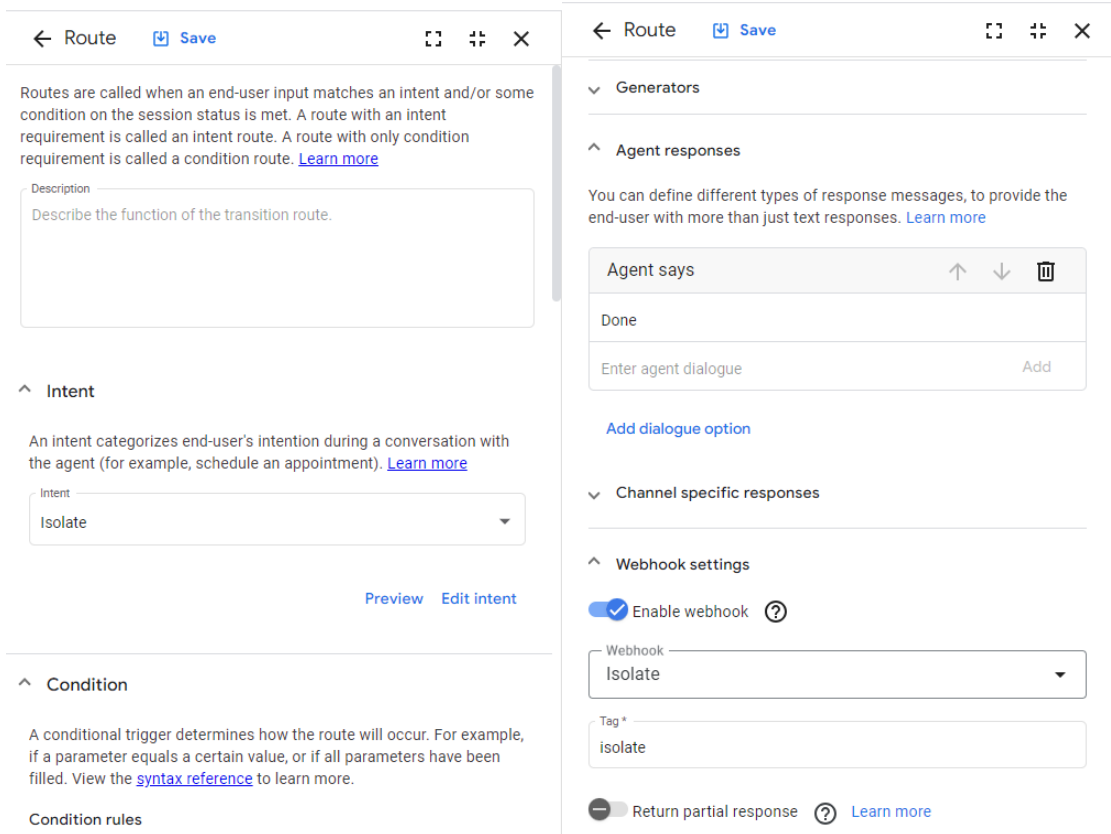


Figure 23: Route definition

The configuration of the Agent for the conversation is completed now, and the next step is to add the agent in the application in order to start the conversation. The code to get the HTML code is located in the “Publish” window of our project as shown in the figure below:

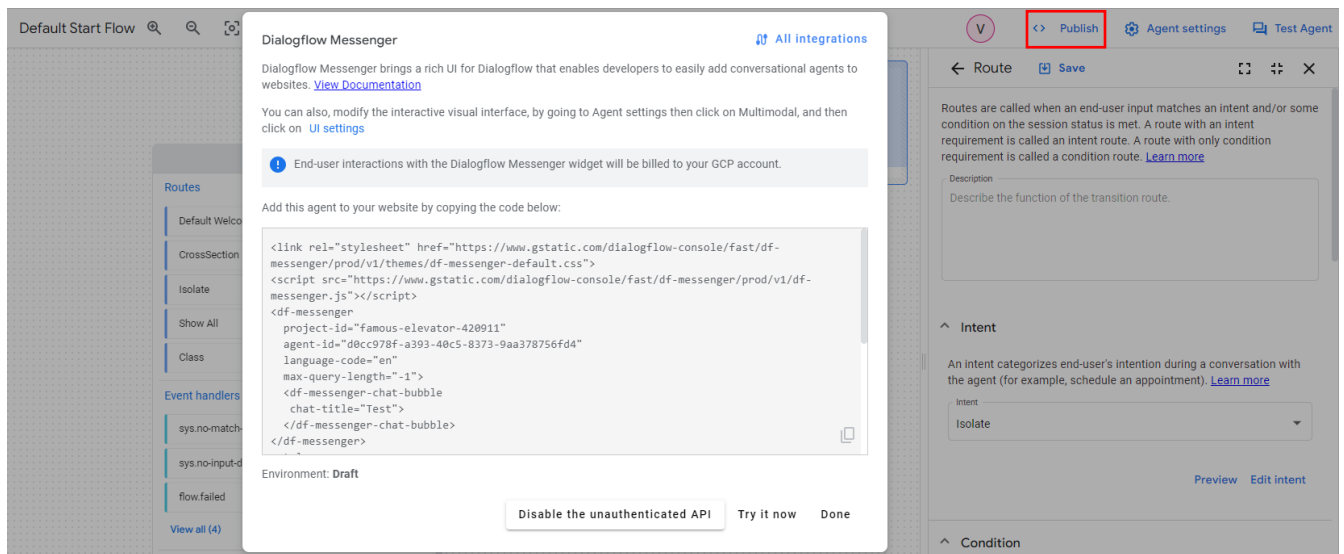


Figure 24: Chatbot UI source code

The Agent can also be directly tested within the platform, under the “Test Agent” section. This tool is useful to analyze errors that can arise during the operation. In the implementation chapter it will be seen how the Chatbot conversation looks like.

4. Prototype

4.1 Detailed Description with Code Snippets of the Web Application

In Figure 23 are shown all the directories and files that make up the web application.

node_modules: This directory contains all the dependencies and modules installed via npm or yarn. These dependencies are listed in the package.json file.

src: This directory holds the source code of the application. The primary file that contains the frontend code to create the application functions is “main.ts”.

package.json: Lists the project dependencies, scripts, and other metadata about the project.

tsconfig.json: The main TypeScript configuration file. It specifies the compiler options and files to include/exclude.

vite.config.ts: Configuration file for Vite, which is a build tool and development server. This file customizes how Vite builds and serves the application.

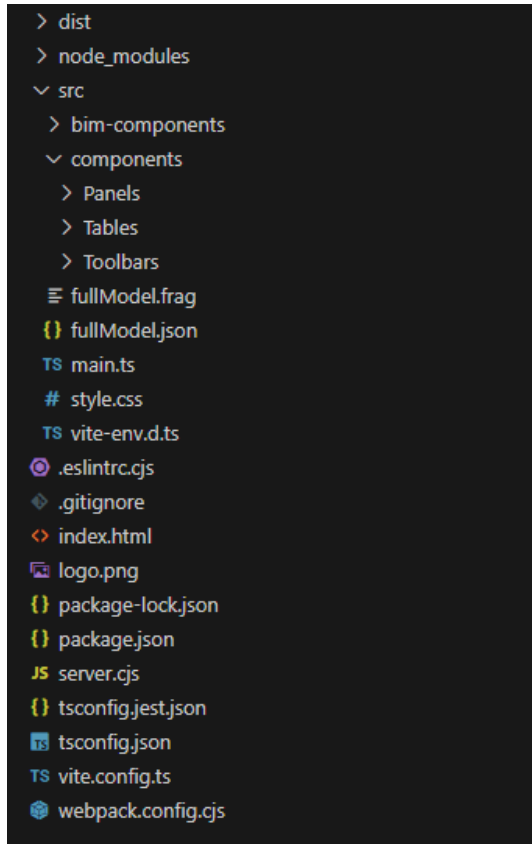
webpack.config.cjs: Configuration file for Webpack, another build tool.

The file named “index.html” is used to create the web page, which uses main.ts as a source file, based on the styles defined in “style.css”.

The file named “server.cjs” and its contents will be explained in the following chapter, when the Chatbot integration will be described.

The first part of “main.ts” involves importing various libraries and components essential for the application. The full project will all the code will be provided at the end. The code lines in this file are quite long, so the most important parts will be explained in detail:

```
import * as THREE from "three";
import * as OBC from "@thatopen/components";
import * as OBF from "@thatopen/components-front";
import * as BUI from "@thatopen/ui";
import * as BUIC from "@thatopen/ui-obc";
import projectInformation from "../components/Panels/ProjectInformation";
import elementData from "../components/Panels/Selection";
import settings from "../components/Panels/Settings";
import load from "../components/Toolbars/Sections/Import";
import help from "../components/Panels/Help";
import camera from "../components/Toolbars/Sections/Camera";
import selection from "../components/Toolbars/Sections/Selection";
import { AppManager } from "../bim-components";
import Stats from "three/examples/jsm/libs/stats.module.js";
```



```
> dist
> node_modules
v src
  > bim-components
  v components
    > Panels
    > Tables
    > Toolbars
  fullModel.frag
  fullModel.json
  TS main.ts
  # style.css
  TS vite-env.d.ts
.eslintrc.cjs
.gitignore
index.html
logo.png
package-lock.json
package.json
server.cjs
tsconfigjest.json
tsconfig.json
vite.config.ts
webpack.config.cjs
```

Figure 25: Project folder contents

Next, the application initializes the UI managers and sets up the core components:

```
BUI.Manager.init();
BUIC.Manager.init();
const components = new OBC.Components();
const worlds = components.get(OBC.Worlds);

const world = worlds.create<
  OBC.SimpleScene,
  OBC.OrthoPerspectiveCamera,
  OBF.PostproductionRenderer
>();
world.name = "Main";
```

The setup for the 3D world involves configuring the scene, camera, renderer, and various other components:

```
world.scene = new OBC.SimpleScene(components);
world.scene.setup();
world.scene.three.background = null;

const viewport = BUI.Component.create<BUI.Viewport>(() => {
  return BUI.html`
    <bim-viewport>
      <bim-grid floating></bim-grid>
    </bim-viewport>
  `;
});

world.renderer = new OBF.PostproductionRenderer(components, viewport);
const { postproduction } = world.renderer;

world.camera = new OBC.OrthoPerspectiveCamera(components);
```

The world grid and resizing functionality ensure that the 3D view is properly scaled and displayed:

```
const worldGrid = components.get(OBC.Grids).create(world);
worldGrid.material.uniforms.uColor.value = new THREE.Color(0x424242);
worldGrid.material.uniforms.uSize1.value = 2;
worldGrid.material.uniforms.uSize2.value = 8;

const resizeWorld = () => {
  world.renderer?.resize();
  world.camera.updateAspect();
};

viewport.addEventListener("resize", resizeWorld);
```

The application loads the 3D model named fullModel. This is the IFC tunnel file that was exported from Civil 3D, and converted by the library in another .frag format. This is the format that the library uses to visualize IFC files. It is a quite fast and efficient way to load IFC files in the web browsers. All .frag files are linked with the .json file which contains all the properties from the original IFC file:

```
const fragments = components.get(OBC.FragmentsManager);
const indexer = components.get(OBC.IfzRelationsIndexer);
const classifier = components.get(OBC.Classifier);
const ifcLoader = components.get(OBC.IfzLoader);
await ifcLoader.setup();
const file = await fetch("src/fullModel.frag");
const data = await file.arrayBuffer();
const buffer = new Uint8Array(data);
const model = await fragments.load(buffer);
const properties = await fetch("src/fullModel.json");
const props = await properties.json();
model.setLocalProperties(props);
```

To build the Alignment visualization features, the library provides these components named: Civil3DNavigator, CivilCrossSectionNavigator, CivilElevationNavigator and CivilPlanNavigator. To visualize the alignment in the 3D view and zoom in at a specific point along it, these lines are used:

```
const navigator = components.get(OBF.Civil3DNavigator);
navigator.world = world;
navigator.draw(model);
const sphere = new THREE.Sphere(undefined, 20);

//Connect point in 3d with cross section
navigator.onHighlight.add(({ point }) => {
  sphere.center.copy(point);
  world.camera.controls.fitToSphere(sphere, true);
  navigator.onMarkerChange.add(({ alignment, percentage, type, curve }) => {
    if (type === "select") {
      const mesh = curve.alignment.absolute[curve.index].mesh;
      const point = alignment.getPointAt(percentage, "absolute");
      crossNavigator.set(mesh, point);
    }
  })
});
```

Next, the link between the horizontal alignment and the cross section of the model is done using CivilPlanNavigator and CivilCrossSectionNavigator. The code creates a synchronization of the marker on the Horizontal alignment and the corresponding cross section at that exact point:

```
const world2D = document.getElementById("Plan-Nav") as BUIC.World2D;
const planNavigator = components.get(OBF.CivilPlanNavigator);
```

```

world2D.components = components;
planNavigator.world = world2D.world;
await planNavigator.draw(model);
const CrossSection = document.getElementById("Cross-Section") as BUIC.World2D;
CrossSection.components = components;

const crossNavigator = components.get(OBF.CivilCrossSectionNavigator);
crossNavigator.world = CrossSection.world;
crossNavigator.world3D = world;

planNavigator.onMarkerChange.add(({ alignment, percentage, type, curve }) => {
  navigator.setMarker(alignment, percentage, type);
  elevationNavigator.setMarker(curve.alignment, percentage, type);
  if (type === "select") {
    const mesh = curve.alignment.absolute[curve.index].mesh;
    const point = alignment.getPointAt(percentage, "absolute");
    crossNavigator.set(mesh, point);
  }
});
planNavigator.onMarkerHidden.add(({ type }) => {
  navigator.hideMarker(type);
});

```

Finally, to implement the Profile view inside the web application, CivilElevationNavigator will be used. The result is a synchronized view of Horizontal alignment, Profile, 3D alignment marker and the cross section at that specific marker location:

```

const elevationNavigator = components.get(OBF.CivilElevationNavigator);
elevationNavigator.world = Profile.world;
await elevationNavigator.draw(model);

planNavigator.onHighlight.add(({ mesh, point }) => {
  const { index, alignment } = mesh.curve;

  const percentage = alignment.getPercentageAt(point, "horizontal");
  if (percentage === null) return;
  const { curve } = alignment.getCurveAt(percentage, "vertical");
  elevationNavigator.highlighter.select(curve.mesh);

  elevationNavigator.setMarker(curve.alignment, percentage, "select");

  if (Profile.world) {
    if (!curve.mesh.geometry.boundingSphere) {
      curve.mesh.geometry.computeBoundingSphere();
    }
    const vertSphere = curve.mesh.geometry.boundingSphere!.clone();

```



```

    vertSphere.radius *= 0.9;
    Profile.world.camera.controls.fitToSphere(vertSphere, true);
  }

  navigator.highlighter.select(mesh);
  const curve3d = mesh.curve.alignment.absolute[index];
  curve3d.mesh.geometry.computeBoundingSphere();
  const sphere = curve3d.mesh.geometry.boundingSphere;
  if (sphere) {
    world.camera.controls.fitToSphere(sphere, true);
  }
});

```

These Civil Navigators components need a 2D world element in order to visualize their content. CSS style was added in order to create a particular interface as will be shown in the results chapter. This is done in the index.html file of the project. The classes are put to hidden when the app starts, in order to control their visibility based on the user's choice:

```

<bim-grid id="app"></bim-grid>
<script type="module" src="/src/main.ts"></script>
<bim-world-2d id="Plan-Nav" class="Plan-Nav hidden"></bim-world-2d>
<bim-world-2d id="Cross-Section" class="Cross-Section hidden"></bim-world-2d>
<bim-world-2d id="Profile" class="Profile hidden"></bim-world-2d>

```

4.2 Integrating Dialogflow CX chatbot with the app

As it was shown previously in the Methodology chapter, in the Publish section of the Dialogflow the source code for the agent is located. This source code looks like this:

```

<link rel="stylesheet" href="https://www.gstatic.com/dialogflow-console/fast/df-messenger/prod/v1/themes/df-messenger-default.css">
<script src="https://www.gstatic.com/dialogflow-console/fast/df-messenger/prod/v1/df-messenger.js"></script>
<df-messenger
  project-id="famous-elevator-420911"
  agent-id="d0cc978f-a393-40c5-8373-9aa378756fd4"
  language-code="en"
  max-query-length="-1">
  <df-messenger-chat-bubble
    chat-title="Test">
  </df-messenger-chat-bubble>
</df-messenger>
<style>
df-messenger {
  z-index: 999;

```

```

position: fixed;
--df-messenger-font-color: #000;
--df-messenger-font-family: Google Sans;
--df-messenger-chat-background: #f3f6fc;
--df-messenger-message-user-background: #d3e3fd;
--df-messenger-message-bot-background: #fff;
bottom: 16px;
right: 16px;
}
</style>

```

It enables developers to easily add conversational agents to the websites. It has a modifiable visual interface, which can be edited directly on the source code or in the Agent settings on the platform.

An important part of implementing this conversational agent is to start a server that hosts the project files. Since Dialogflow deals with HTTP request, a server is needed to communicate with the external services. A running server can maintain persistent connections, such as Server-Sent Events (SSE), which are necessary for real-time updates to clients based on actions triggered by Dialogflow. The server handles incoming data from Dialogflow and can process, store, or manipulate this data as required. It can also send specific responses back to Dialogflow based on this data.

The file that starts the server is named “server.cjs” and it looks like this:

```

const express = require('express');
const path = require('path');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json());
// Serve static files from the 'dist' directory (where bundle.js is generated)
//app.use(express.static(path.join(__dirname, 'dist')));
app.use('/dist', (req, res, next) => {
  res.setHeader('Content-Type', 'application/javascript');
  next();
}, express.static(path.join(__dirname, 'dist')));
// Serve index.html from the root folder
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
// Array to store SSE clients
let sseClients = [];
app.use('/src', express.static(path.join(__dirname, 'src')));
// Route to handle SSE connections
app.get('/events', (req, res) => {
  // Set the headers for SSE
  res.setHeader('Content-Type', 'text/event-stream');
  res.setHeader('Cache-Control', 'no-cache');
  res.setHeader('Connection', 'keep-alive');

```

```

// Send an initial response to keep the connection open
res.write("\n");

// Add the response object to the SSE clients array
sseClients.push(res);

// Handle client disconnect
req.on('close', () => {
  // Remove the disconnected client from the array
  sseClients = sseClients.filter(client => client !== res);
});
});

// Handle POST requests at '/CrossSection'
app.post('/CrossSection', (req, res) => {
  // Send 'Action1' to all connected clients via SSE
  const actionData = {
    action: 'Action1'
  };
  sendActionToClients(actionData);

  // Respond to Dialogflow webhook with a JSON object
  const jsonResponse = {
    message: 'Webhook received the message.'
  };
  res.status(200).json(jsonResponse);
});

// Handle POST requests at '/Isolate'
app.post('/Isolate', (req, res) => {
  // Send 'Isolate' action to all connected clients via SSE
  const actionData = {
    action: 'Isolate'
  };
  sendActionToClients(actionData);

  // Respond with a JSON object
  const jsonResponse = {
    message: 'Isolate action sent to clients.'
  };
  res.status(200).json(jsonResponse);
});

// Handle POST requests at '/ShowAll'
app.post('/ShowAll', (req, res) => {

```

```

// Send 'ShowAll' action to all connected clients via SSE
const actionData = {
  action: 'ShowAll'
};
sendActionToClients(actionData);

// Respond with a JSON object
const jsonResponse = {
  message: 'ShowAll action sent to clients.'
};
res.status(200).json(jsonResponse);
});
// Handle POST requests at '/Class'
let storedClass = "";

// Endpoint to store the class from the client-side
app.post('/Class', (req, res) => {
  const Class = req.body.Class;

  if (!Class) {
    return res.status(400).json({ error: 'Class is required' });
  }

  storedClass = Class;
  //console.log('Stored class:', storedClass);

  res.json({ message: 'Class stored successfully' });
});

// Endpoint for Dialogflow webhook
app.post('/ClassWebhook', (req, res) => {
  if (!storedClass) {
    return res.status(400).json({ error: 'No Class available' });
  }

  const jsonResponse = {
    fulfillmentResponse: {
      messages: [
        {
          text: {
            text: ['The class of this element is: ${storedClass}']
          }
        }
      ]
    }
  }
});

```

```

};

console.log('Responding to Dialogflow with:', jsonResponse);
res.json(jsonResponse);
});

// Function to send SSE action data to all clients
function sendActionToClients(actionData) {
  sseClients.forEach(client => {
    client.write(`data: ${JSON.stringify(actionData)}\n\n`);
  });
}
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

```

The server is built using Node.js and the Express framework. Its functionality can be summarized as follows:

Static File Serving:

- Serves static files (e.g., bundle.js) from the 'dist' directory. “bundle.js“ is the compiled file of “main.ts”. This is a common practice to serve files for web applications from TypeScript to JavaScript.
- Serves index.html from the root directory.
- Serves static files from the 'src' directory.

Server-Sent Events (SSE):

- Handles SSE connections to send real-time updates to clients.
- Maintains a list of connected SSE clients and handles client disconnections.

Routes for Dialogflow Integration:

- Several POST endpoints (/CrossSection, /Isolate, /ShowAll, /ClassWebhook) handle specific actions from Dialogflow. These actions are broadcast to all connected clients (“main.ts”) via SSE. The endpoints are the Webhooks that were created in the Dialogflow platform as shown previously in Figure x.

In order to provide a connection and an action from the “main.ts” code (the uncompiled code of bundle.js) when a POST request is made to server via the Dialogflow agent, these lines are added to the application:

```

// The chatbot part interaction with the viewer
const exportButton = document.createElement('button');
exportButton.innerText = 'Export Cross Section to DXF';
exportButton.onclick = () => exportCrossSectionToDXF(crossNavigator);

```

```
const eventSource = new EventSource('/events');
eventSource.addEventListener('message', (event) => {
  const data = JSON.parse(event.data);
  console.log('Received message from server:', data.message);

  // Handle the response data here
  switch (data.action) {
    case 'Action1':
      exportButton.click();
      break;
    case 'Isolate':
      Isolate();
      break;
    case 'ShowAll':
      ShowAll();
      break;
    default:
      console.log('Unknown action:', data.action);
  }
});

eventSource.addEventListener('error', (error) => {
  console.error('EventSource error:', error);
});
```



The functionalities of this snippet can be described as follows:

1. Setting Up the Client-Side Event Listener

The client-side script sets up an EventSource to listen for events from the server at the /events endpoint. This creates a persistent connection to the server, allowing the server to push updates to the client in real-time.

2. Handling Server-Sent Events

The client registers an event listener for messages from the server.

Data Parsing: The event listener parses the received data using `JSON.parse(event.data)`.

Action Handling: Based on the action field in the received data, the script performs different actions:

- Action1: Simulates a button click to export the cross-section to DXF.
- Isolate: Calls the Isolate function.
- ShowAll: Calls the ShowAll function.

3. Handling Errors

The client also registers an error event listener

The following code parts are responsible for the Agent fulfillment response in the conversation. The fulfillmentResponse has a defined json structure which Dialogflow expects when a post request is made, and the given response is in text. In this case the text is made of: [‘The class of this element is: \${storedClass}’], where \${storedClass} is the parameter extracted from the client side which contains the properties of the IFC file. In this way, many other webhooks can be created following a similar structure and using a different parameter based on the IFC properties.

The class parameter is sent from the client side to the server via a POST request at /Class using fetch function. It is extracted from propertiesTable.tsv which contains all the properties of the selected 3D element. This part of the code is located under “src/components/Panels/Selection.ts” :

```
if (Class) {
  console.log('Class:', Class);

  // Send the description to the server
  fetch('/Class', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ Class: Class }),
  })
  .then(response => response.json())
  .then(data => console.log('Server response:', data))
  .catch(error => console.error('Error:', error));
} else {
  console.log('Class not found in propertiesTable.tsv');
}
```

And the server-side code stores the parameter like this:

```
// Endpoint to store the description from the client-side
app.post('/Class', (req, res) => {
  const Class = req.body.Class;

  if (!Class) {
    return res.status(400).json({ error: 'Class is required' });
  }

  storedClass = Class;
  //console.log('Stored class:', storedClass);

  res.json({ message: 'Class stored successfully' });
});
```

Now that the class parameter is stored in the server, it can be sent anytime to any client that POSTs a request to retrieve that information. In this case, it is enough for a simple Dialogflow webhook to activate a POST request that sends back a fulfillment response with the stored parameter as the following code snippet:

```
// Endpoint for Dialogflow webhook
app.post('/ClassWebhook', (req, res) => {
  if (!storedClass) {
    return res.status(400).json({ error: 'No Class available' });
  }

  const jsonResponse = {
    fulfillmentResponse: {
      messages: [
        {
          text: {
            text: ['The class of this element is: ${storedClass}']
          }
        }
      ]
    }
  }
}]
```

In summary, the provided codes enable the user to interact with the Dialogflow CX chatbot and do the following:

- Send queries that control the visibility of 3D elements in the view (e.g Isolate, Show All)
- Export the Cross Section as a DXF file
- Extract property information from the IFC model properties (e.g. IFCClass)

4.2 User Interface Design

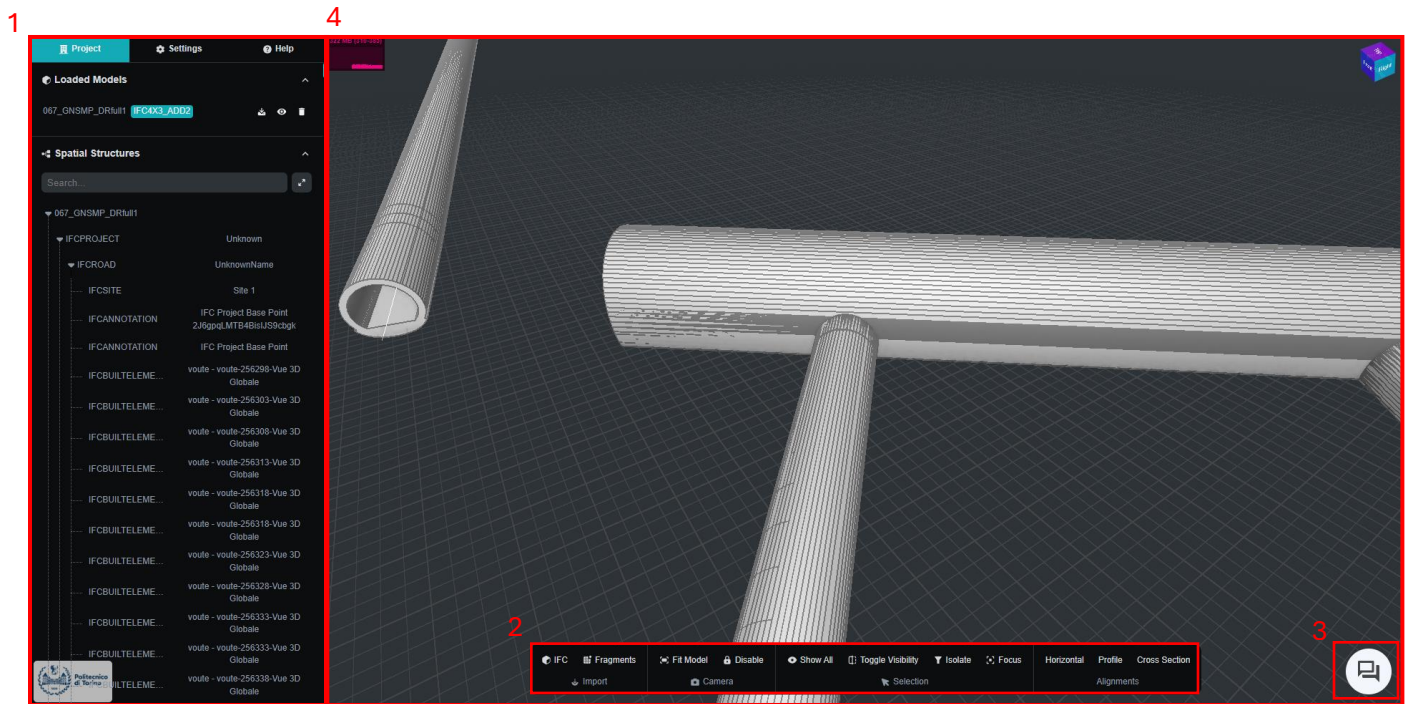


Figure 26: User Interface of the application

1 - The left Panel

This panel on the left side of the screen contains:

- **Project Section:** Allows users to manage project-related settings and details.
- **Settings Section:** Provides access to various application settings and preferences.
- **Help Section:** Offers help resources or documentation for the application.
- **Loaded Models Section:** Displays a list of models that are currently loaded in the application. Each model can be expanded to show more detailed information about its structure.
- **Spatial Structures:** Lists the hierarchical structure of the loaded models, such as IFCPROJECT, IFCROAD, IFCBRIDGE, etc. Users can explore different components and sub-components of each model.
- **Search Bar:** Allows users to search for specific elements within the loaded models or spatial structures.

2 - Toolbar

This toolbar at the bottom of the screen contains:

- **Import Button:** Allows users to import new models or data into the application.
- **Camera Buttons:** Provides options to adjust the camera view and settings.

- **Toggle Visibility Button:** Enables users to show or hide specific elements in the model.
- **Show All Button:** A quick action to display all elements in the model.
- **Isolate Button:** Focuses on a specific element while hiding others.
- **Focus Button:** Centers the camera on a selected element.
- **Horizontal Button:** Opens the horizontal alignment panel.
- **Profile Button:** Opens the Profile panel of the model.
- **Cross Section Button:** Allows users to view cross-sections of the model based on alignment marker location.

3 – Chatbot (Dialogflow CX Agent)

It opens the Conversational Agent window that allows the interaction between the user and the application using natural language. When clicked it looks like this:

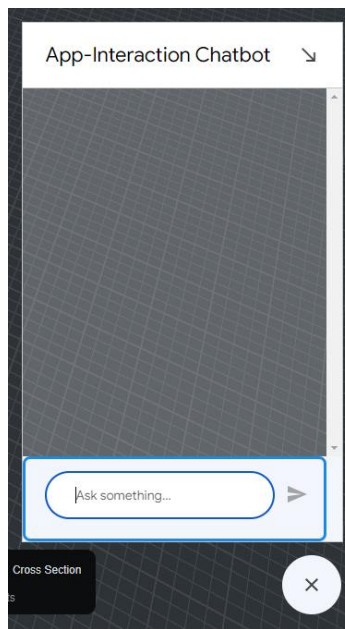


Figure 27: Chatbot UI component

4 - Main Viewport

This is the central part of the screen where the 3D model is displayed:

- **3D Model Display:** Shows the current view of the loaded 3D models. Users can interact with the model through zooming, rotating, and panning to inspect different aspects of the model.
- **Model Elements:** Displays the geometrical details of the model elements. Users can click on different parts of the model to get more information or perform specific actions.

The viewport contains a panel (top left) that measures the performance of the app, in terms of RAM, frames per second, and milliseconds and a view cube that shows the orientation of the scene (top right).

When an element is clicked another panel with its properties shows up:

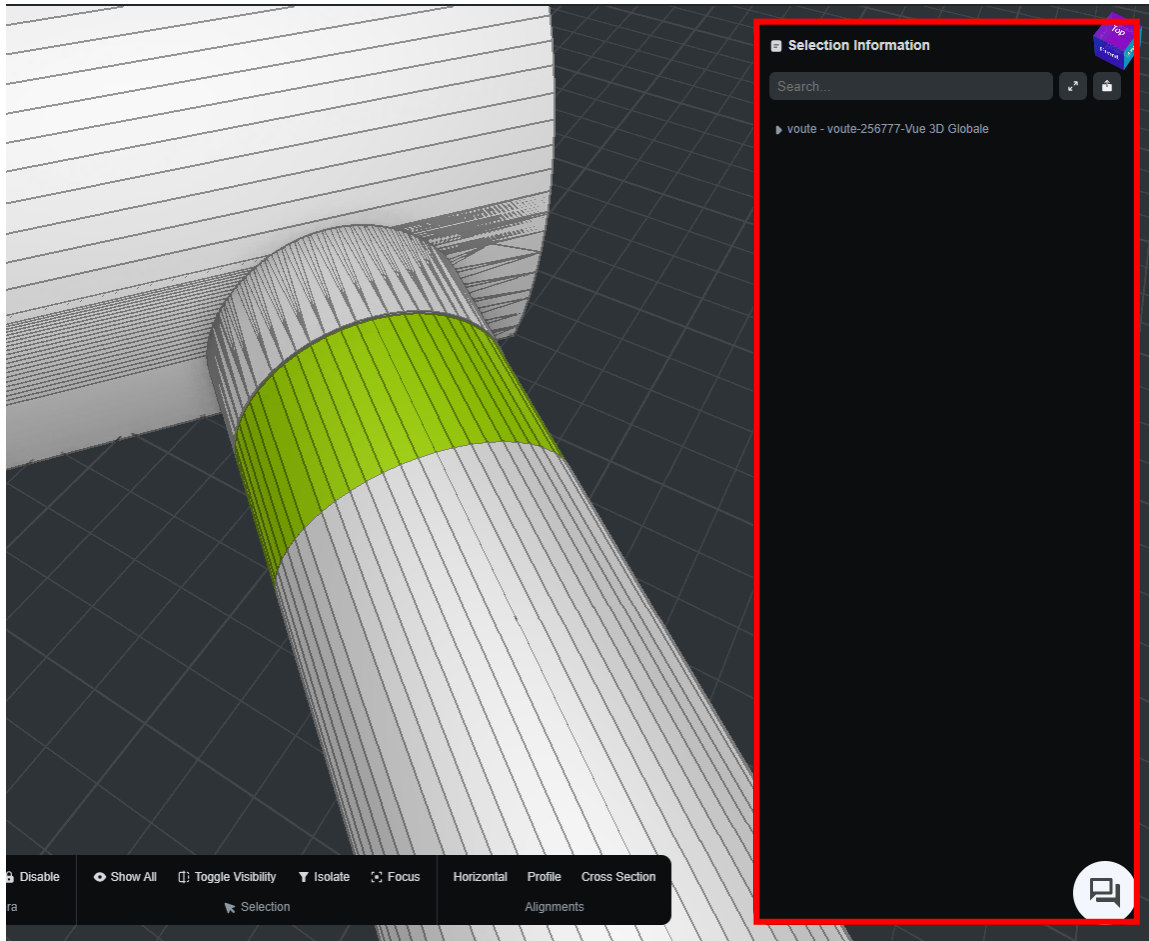


Figure 28: Properties Panel

This panel has a search bar that users can use to search for specific properties, and an export button (top right) that allows users to export the properties as a tsv format.

5. Results

5.1 Outcomes of the Implementation

The result of the web development of the IFC viewer for the case study model looks like this:

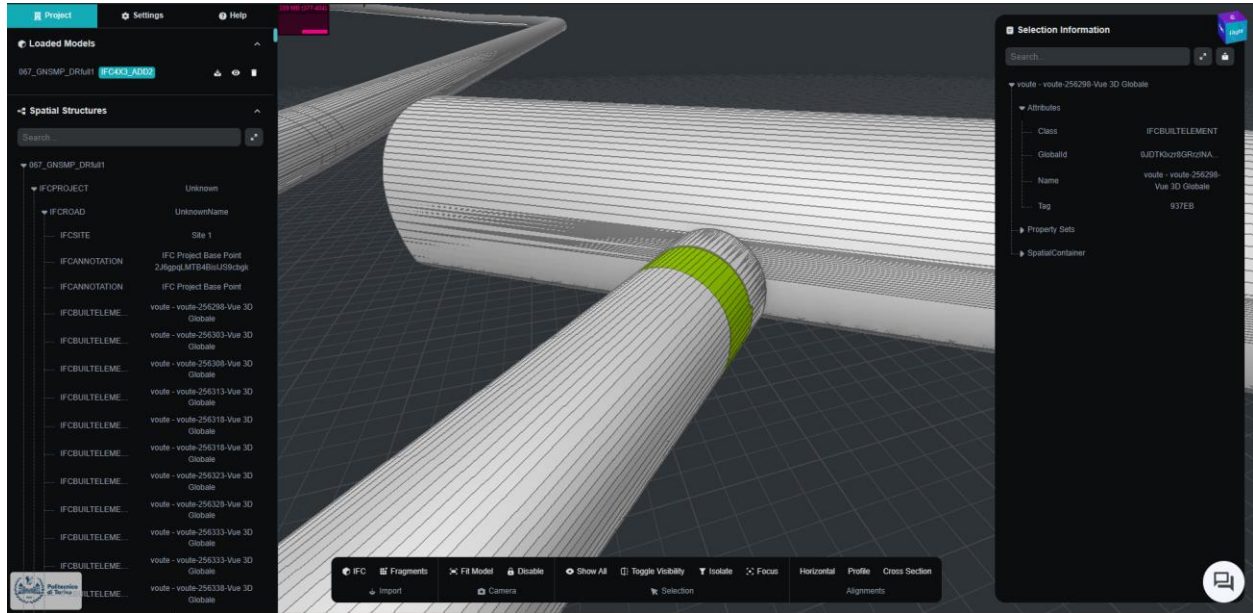


Figure 29: The Tunnel Model loaded in the Application

The user is able to visualize the 3D geometry and apply different visibility functions, analyze the Model tree, analyze the selected element's properties and adjust different styles to the app in the settings panel. The user can also see how much memory the model is consuming, how fast are the frames loaded etc. Related to the Alignment visualization, the added components look like this:

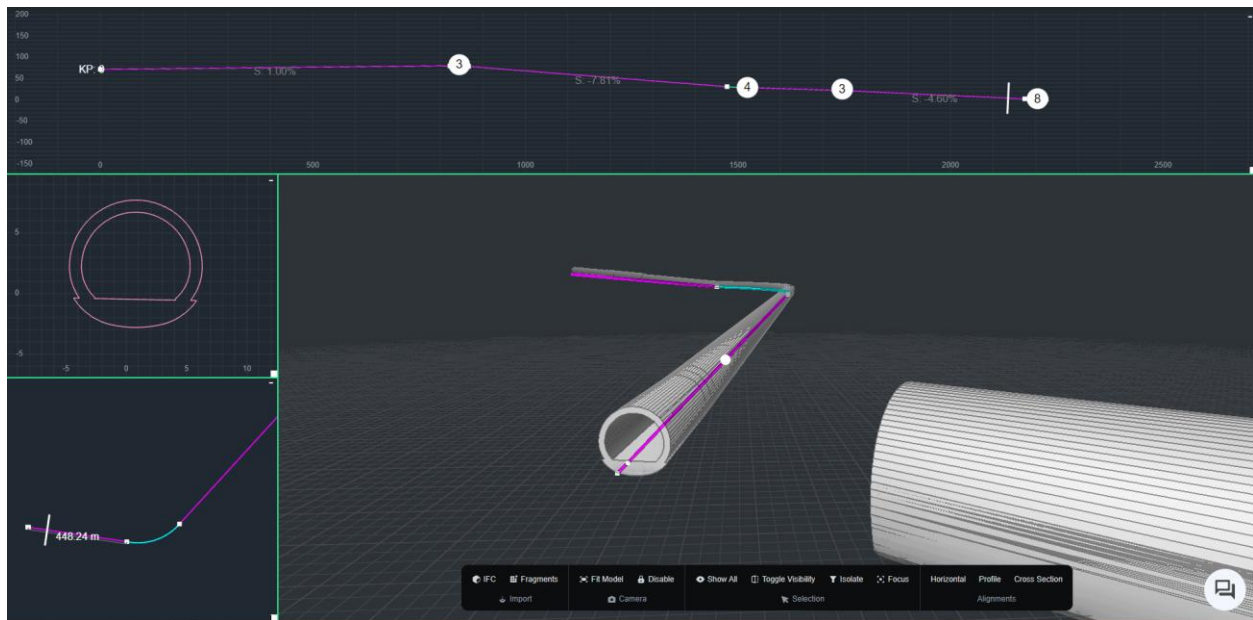


Figure 30: Visualized Alignments in the Application

There are three panels that can be opened but clicking the respective buttons in the toolbar. Firstly, the user can visualize the horizontal alignment in the bottom left panel. Each part of the alignment like a line, curve or clothoid shows the length of the segment of the radius if it is a curve. The marker in panel is synchronized with the 3D alignment, the profile marker and the cross section at that point. In this case, that position of the markers shows the cross section of the model at that point.

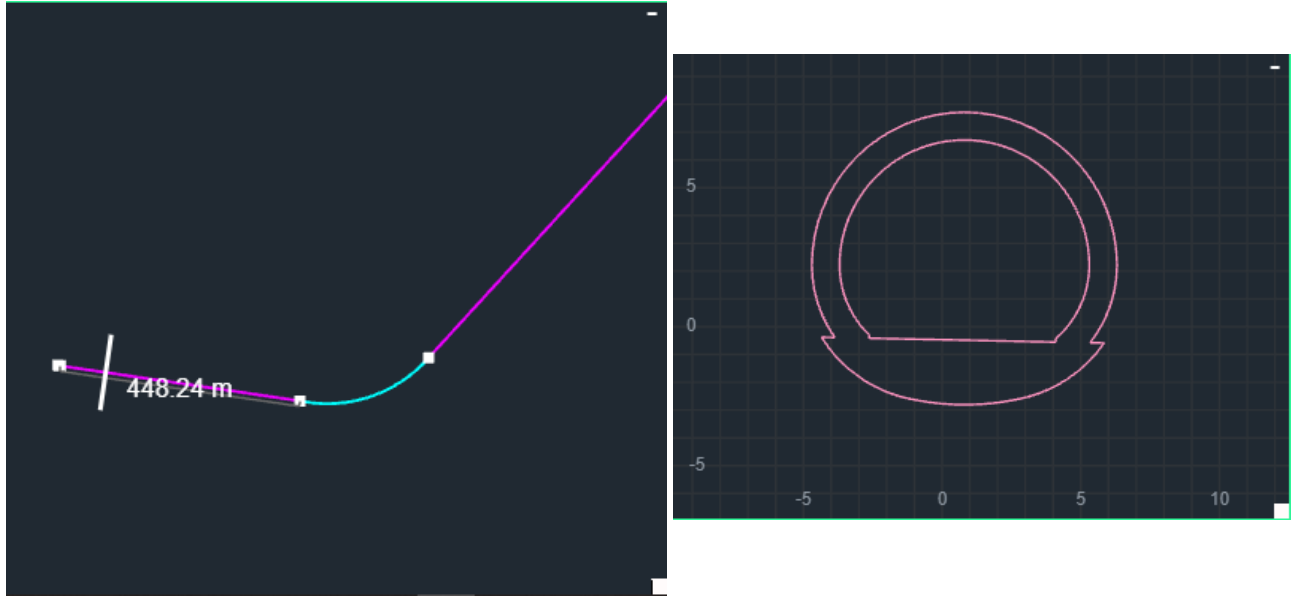


Figure 31: Horizontal Alignment and Cross-Section Panel

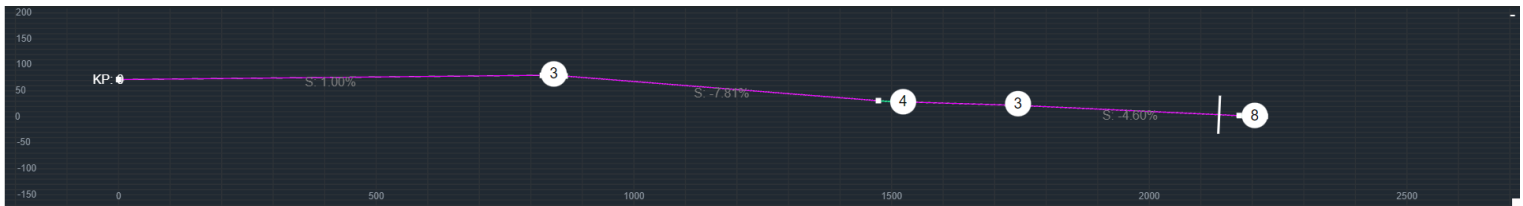


Figure 32: The Profile View Panel

The Profile panel (Figure 30) shows information related to the slope, height at specific stations and the position of the marker along the alignment. All of the three panels are easy to navigate, meaning that the user can zoom in and out many times and click over the alignment for a fast camera focus.

The results of the Chatbot implementation include include 4 actions:

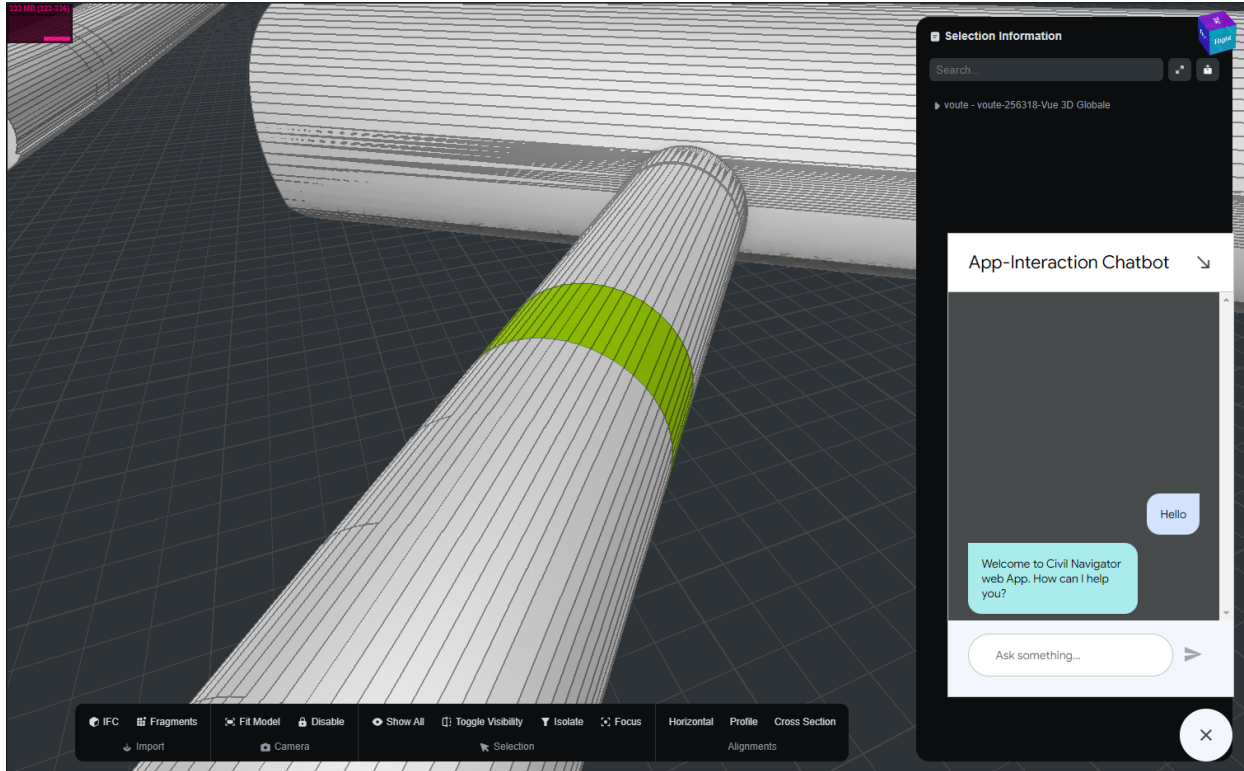


Figure 33: Chatbot Agent Start

The Conversational Agent starts automatically when the page is loaded, so phrases like “Hello” or “Hi” do not have any specific options. They are used in this case to show that the responses are customizable, and the user may choose different approaches for the Agent Start.

1 – Isolating the selected element of the model

The user will ask the Agent to Isolate the selected element shown in Figure 34 by typing something like “Can you isolate this element?”. The result looks like this:

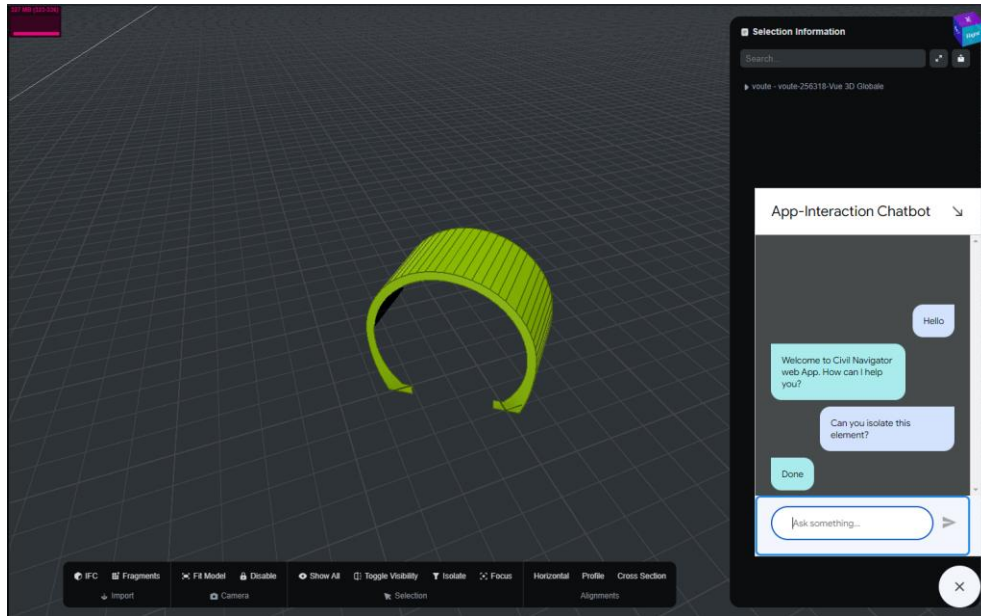


Figure 34: Isolate function executed by Chatbot

After the action is completed, the Agents responds with “Done”. In case of any error, it would have responded with “Webhook error” which means Dialogflow was not able to make a POST request to the server.

2 – Showing all the hidden elements in the 3D view

The user will ask the Agent to Isolate the selected element shown in Figure 33 by typing something like “Show all the hidden elements”.

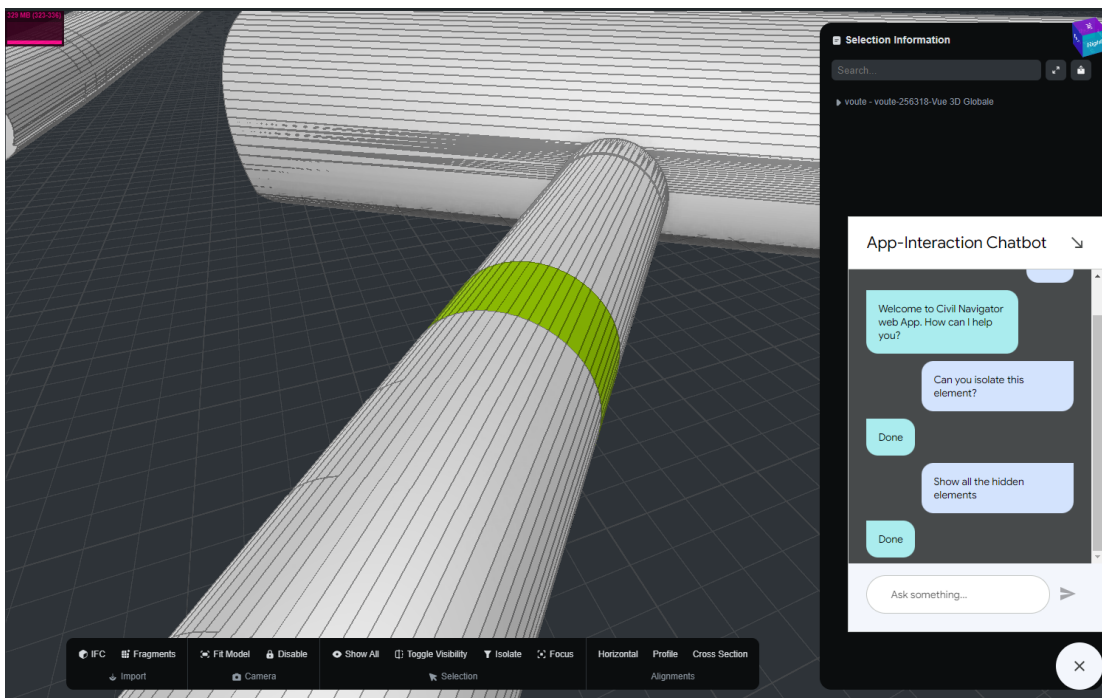


Figure 35: "Show all hidden elements" function executed by Chatbot

3 – Asking what IFC class does the element belong to?

The IfcClass entity is important because it defines the specific type of an object or element within a model. The user can ask the Agent to know what is the IFC class of the selected element and it will respond back with the correct answer based on its properties. (Figure 34)

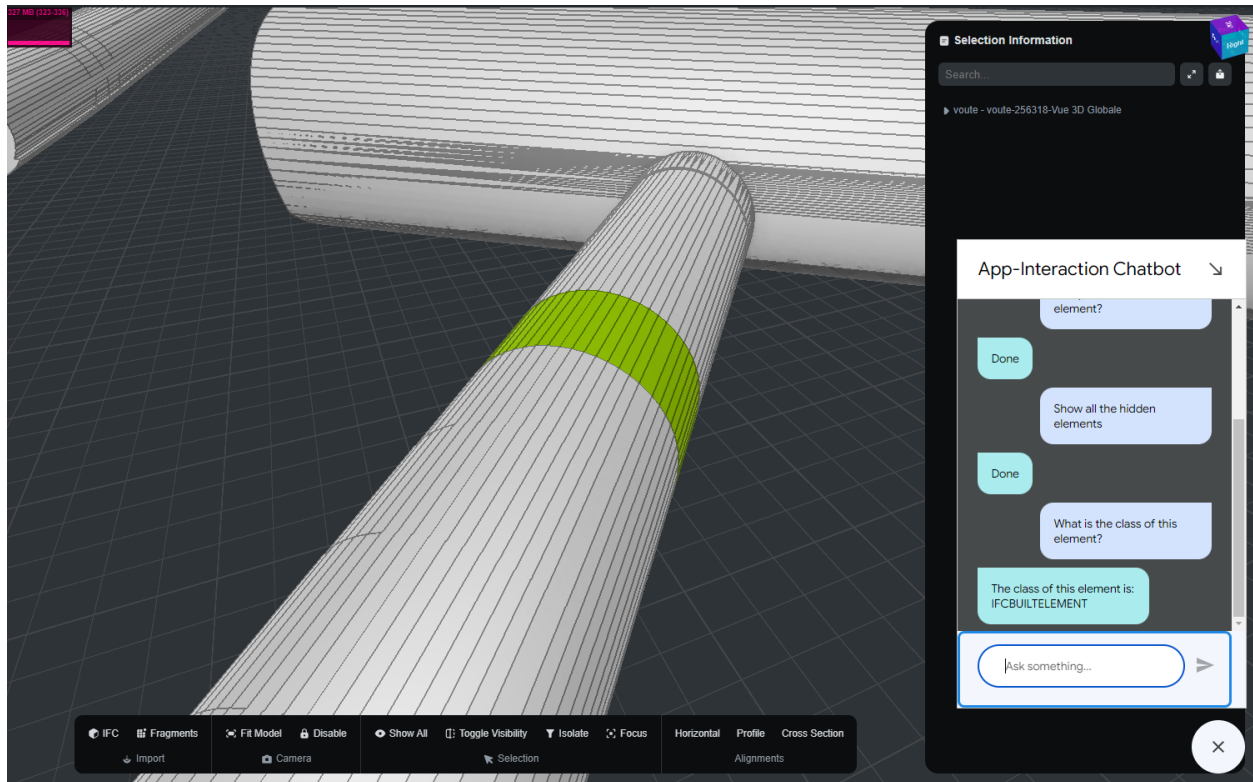


Figure 36: Chatbot response with the IFC class of the selected element

This element is an IFCBUILTELEMENT. The answer will vary for different elements because the Agent sends back the selected element's properties from the server via the fulfillment text of the POST request.

4 – Exporting the Cross section panel as a DXF file

A DXF (Drawing Exchange Format) file can be useful for the user to analyze the cross section geometry in a CAD software, or for documentation purposes. The chatbot is able to trigger a function that will download the Cross Section panel as a DXF with just a text line. (Figure 35)

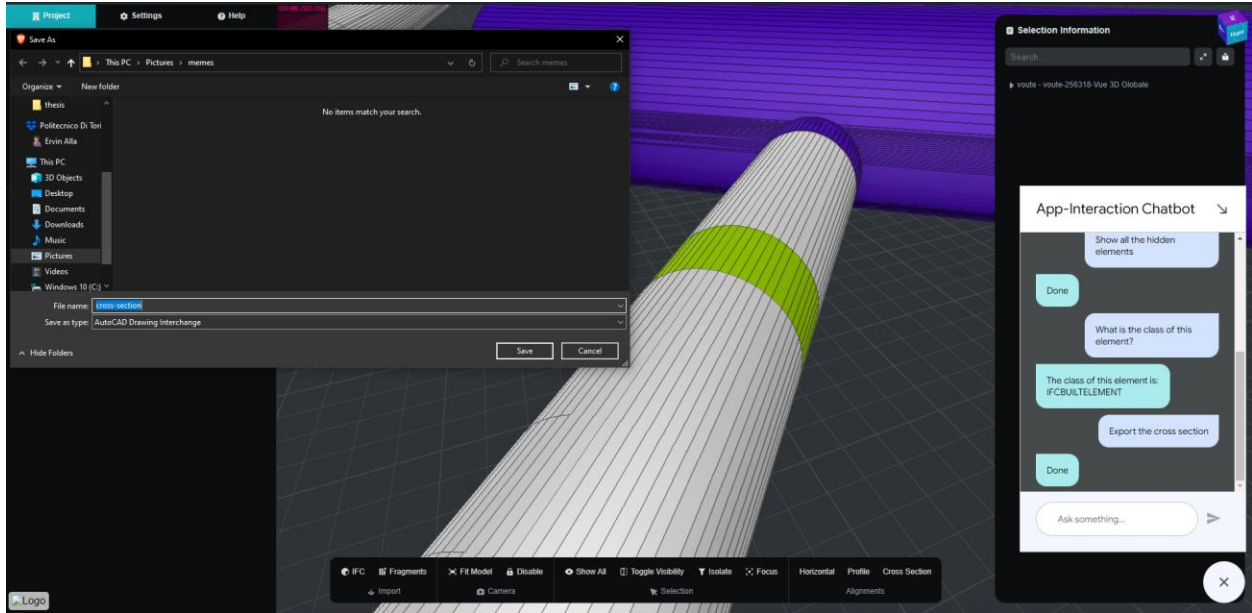


Figure 37: Exported Cross Section Panel as DXF file

The browser automatically opens a “Save as” window for the exported file. In this case the cross section is connected to the alignment marker, but it can be further developed to export geometries that are not part of the alignment too.

5.2 Performance Benchmarks

In the web application a “culler” function was implemented in order to improve the performance in the browsers. Culler functions selectively process or render only the visible parts of the 3D model, skipping over non-visible or irrelevant portions. By rendering only these parts, it significantly reduces the rendering load on the GPU. When navigating a large building model, culler functions ensure only the relevant portions of the building are loaded into memory, rather than the entire model, which conserves RAM and improves performance.

Loading this IFC model inside the application uses about 330 MB of RAM and loads at around 20 FPS. The hardware components of the PC where this app is loaded affect those statistics.



Figure 38: Performance Indicator Panels

A better and more powerful PC would provide a smoother navigation experience in the application. A good graphic card can easily upgrade the FPS up to 60.

6. Conclusion and Future Development

As described in the “Results” chapter, the application provides a fully functional IFC Viewer equipped with basic tools for visualization and navigation, in addition to advanced features like alignment visualization and a Conversation Agent integration with the app. This viewer is highly beneficial for various professionals, particularly a BIM Manager working on infrastructure projects.

The IFC Viewer’s primary functionality allows users to visualize and navigate large infrastructure models effectively. Basic tools such as pan, zoom, and rotate enable intuitive interaction with extensive models, offering a clear view of different parts. Section cuts and layers allow users to isolate and examine specific sections in detail, which is essential for thorough inspections and design reviews.

The alignment visualization feature is a standout, which allows users to see the precise positioning and alignment of infrastructure elements like roadways, bridges, and tunnels, ensuring everything aligns correctly according to design specifications. This feature eliminates the need for additional, and sometimes costly, software like Autodesk Civil 3D which makes the application a powerful alternative for visualization.

The Conversational Agent (Chatbot), which is powered by natural language processing, further improves the user experience. This chatbot translates simple natural language commands into actions within the application. For instance, users can simply ask, "Isolate this element," and the chatbot will execute the command. This feature makes the interface accessible even to those with minimal technical training, which allows to broaden the user base and improve overall productivity.

Use Case Scenario: BIM Manager on an Infrastructure Project

Imagine a BIM Manager who oversees the construction of a new highway, complete with bridges, tunnels, and interchanges. This project spans several kilometers, and the BIM Manager is responsible for auditing IFC models delivered by different contractors. The size and complexity of these models pose significant challenges, requiring powerful visualization tools to navigate and inspect them effectively. Budget constraints make it impractical to purchase expensive software licenses, so a free IFC viewer that can handle large models becomes invaluable.

Using the developed application created by open-source tools, the BIM Manager can load and navigate the entire highway model with ease. The basic tools allow for smooth exploration of different parts of the infrastructure models using a web browser. During a coordination meeting, the BIM Manager can visually demonstrate that all elements of the model across its length span, thanks to the alignment visualization feature.

The BIM Manager can read various properties of model elements directly within the viewer and export cross sections at a specific point along the alignment for detailed analysis. For instance, during a review meeting, they might need to present the structural details of a tunnel section. The application enables them to export the necessary cross sections and further analyze their components in a CAD software.

The Conversation Agent integration simplifies interaction with the model. The BIM Manager can write natural language commands, such as "Show me the class of the selected IFC element," and the chatbot will display the requested information. This feature is useful for new team members who are unfamiliar with the software. Instead of navigating through complex menus, they can ask the chatbot to isolate specific elements and the chatbot will automatically execute the task.

The application's accessibility via a web browser is a major advantage. This eliminates the need for complex installations and makes it easy for project stakeholders to access the model from various locations and devices. For example, the BIM Manager might be on-site and need to verify some details. Using a tablet, they can access the application through a web link, navigate to the necessary part of the model, and use the chatbot to quickly find the required information—all without needing to install any software.

Limitations

One of the primary limitations lies in the technical performance of the application when handling extremely large or highly detailed models. The application is designed to manage extensive infrastructure models, but there may be performance issues such as slow rendering times or lag when working with very large datasets or highly complex geometries. This could impact the user experience for projects that require real-time collaboration and quick access to model information.

The application includes the main tools for visualization and navigation, but it lacks some advanced features available in other specialized software. For example, more sophisticated analysis tools, detailed simulation capabilities, or reporting functions are not integrated into this viewer. Users who require these advanced functionalities might still need to rely on additional software, which could limit the overall utility of this application in certain contexts.

As the application runs in a web browser, its performance can be influenced by the browser's capabilities and the device's hardware. Users with older or less powerful devices may experience a weak performance. Additionally, although the application is designed to be cross-browser compatible, there may be inconsistencies or issues in certain browsers that could affect usability.

Given that the application is web-based, data security and privacy are critical concerns. Users must trust that their data is handled securely when dealing with sensitive project information. Developing and maintaining this web application requires a diverse set of technical skills such as: web development, 3D graphics and visualization, BIM and IFC knowledge, Natural Processing Language, Backend Development, security and performance optimization.

This application can be easily improved in several areas:

1.Enhanced Performance

In this project a geometry data format called “Fragments” was used which represents the mesh of the object from the library three.js. However, That Open libraries offer another way to load the IFC models called “Tiles”. This enables IFC models with hundreds of Megabytes or some Gigabytes of data to load in seconds in the browser, increasing the loading speed exponentially.

2. Advanced Analytical Tools

Again, the libraries used for this app development offer a much wider range of features which were not all implemented. Also, other IFC related libraries like ifcopenshell can be explored in order combine them both in a single application. That would make the application able to handle both visualization and authoring actions. For example, features such as:

- structural analysis
- clash detection

- 4D simulation
- BCF

3. Improved Chatbot Capabilities

In this application some basic functions were implemented in order to show how NPL can be used to interact with an IFC viewer. Given the defined steps to create the Intents, many other functionalities can be implemented in order to have a fully capable Chatbot that can answer almost anything that the application can do. This could also involve integrating more advanced AI and LLMs such as ChatGPT, Google Gemini, etc.

4. Offline Functionality

Developing offline functionality would further increase the application's utility in cases when internet connectivity might be unreliable. This could involve creating a desktop version of the viewer that can be used without an internet connection.

5. Enhanced Security Features

Future improvements could include more robust security measures when accessing the app via the browser. This might involve implementing advanced encryption methods, secure access controls, and auditing capabilities to ensure that user data is always protected. For example, authentication connections can be used in order to prevent unauthorized users from accessing the web page.

6. Integration with other tools

Future work could explore integration with other popular BIM tools and platforms. One way to do it is by creating APIs or adopting existing standards for interoperability, where the application could facilitate smoother data exchange and collaboration across different systems. This would enhance its utility in multi-tool workflows and broaden its adoption in the industry. An example might be integration with PowerBI, Cesium for geospatial data, or creating an extension in Microsoft Teams that loads the viewer.

This project has been published as a Repository in GitHub with the following link:

https://github.com/ervinalla99/Tesi_Politecnico

Once the repository is cloned in a local computer, the user can install the dependencies by running the command “npm install” inside the directory in order to install all the modules and packages.

It can be further developed in the areas mentioned above but also using other technologies.

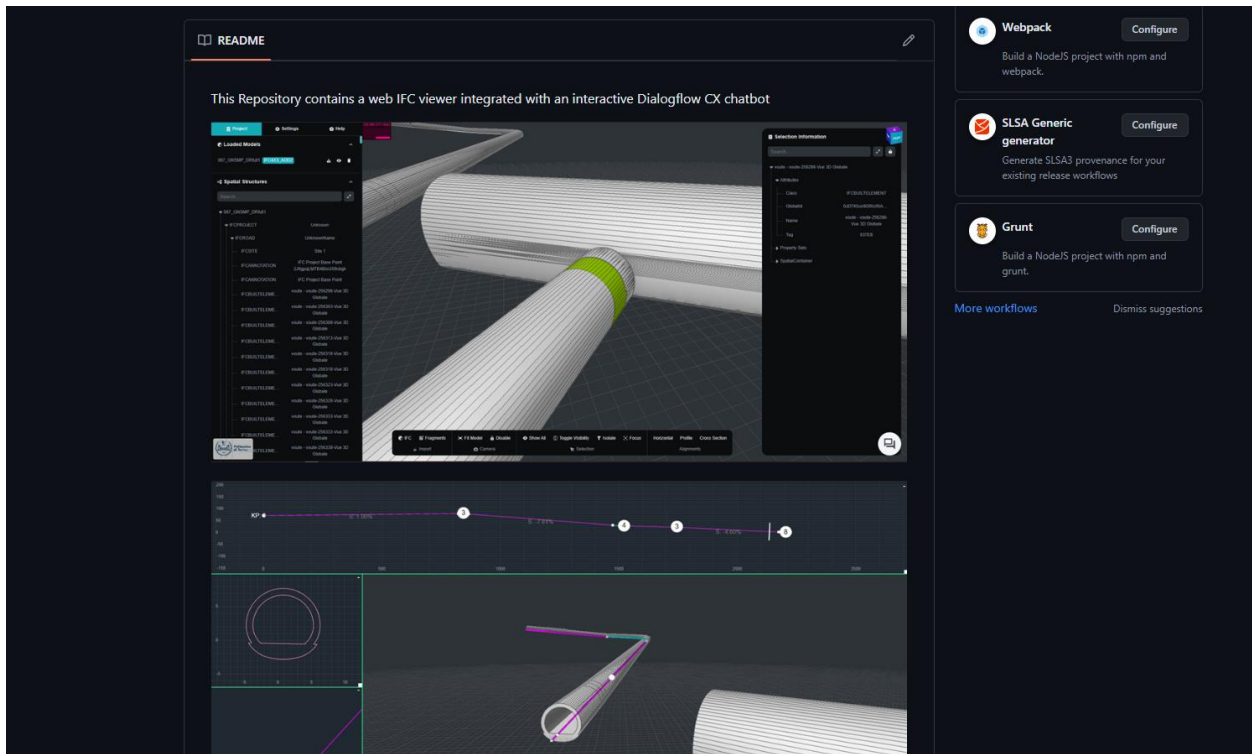
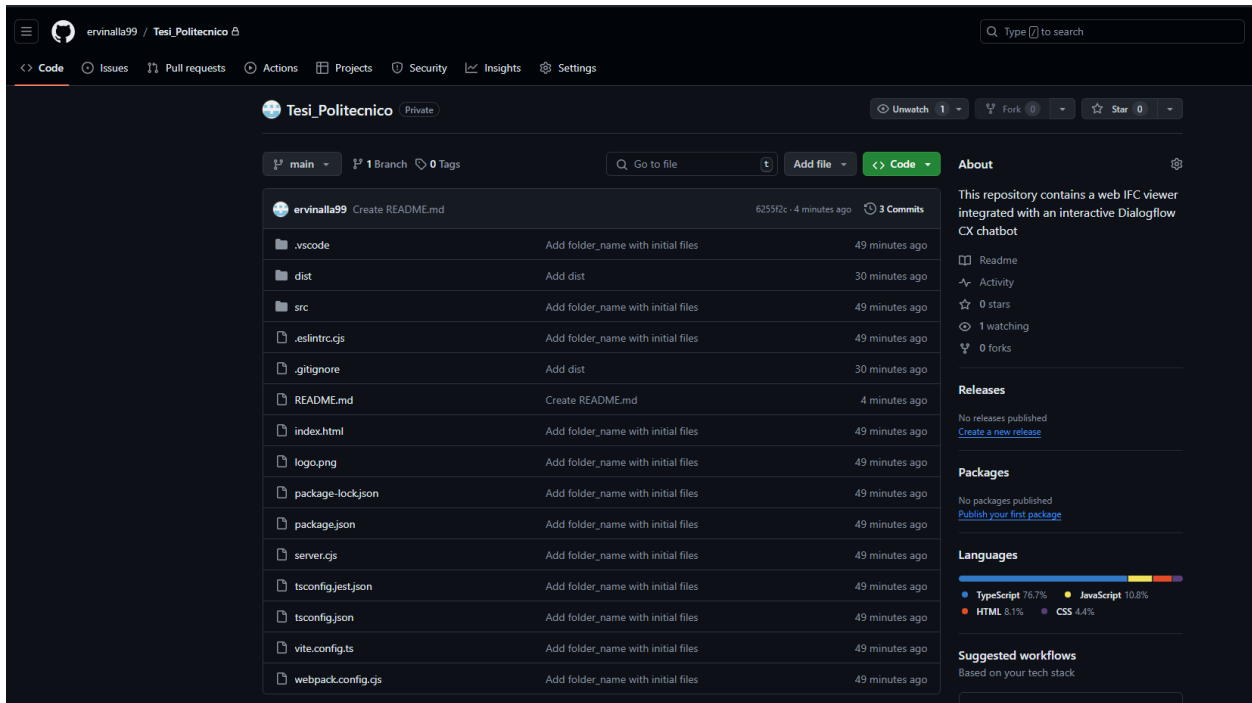


Figure 39: GitHub Repository of the Project

This thesis has detailed the development and implementation of an IFC Viewer application that combines advanced visualization tools with a Conversational Agent that uses NLP. This study's aim was to improve accessibility and usability in the context of Building Information Modeling (BIM) for infrastructure projects. The summary of the key findings from this study is provided below:

- Functionality and Features: The IFC Viewer application integrates advanced tools for the visualization and navigation of IFC infrastructure models, including alignment visualization which is critical for infrastructure projects. Additionally, the application incorporates a natural language processing chatbot that allows users to interact with the model using intuitive commands.
- Accessibility and Usability: The application is easily accessible because of its web-based nature. This allows users to connect with the app from any device with an internet connection and eliminates the need for installing and maintaining software on multiple devices. The integration of the Conversational Agent provides a simple user interface which enables non-technical users to interact with the model effectively. This feature is beneficial for stakeholders who may not have in-depth technical knowledge but need to engage with the project data.
- Cost-Effectiveness: The application makes advanced BIM tools available to a broader audience by providing a free and accessible alternative to expensive proprietary software solutions. This is especially beneficial for projects with limited budgets, such as small firms or public sector projects, where high software costs can be a barrier to effective BIM implementation. The application's open-source nature encourages further development and customization without experiencing additional costs.

The contribution of this study to BIM and project management in infrastructure projects is as follows:

1. **Inclusive Use of Innovation**: The development of a free, open-source, completely Web-based IFC Viewer democratizes access to sophisticated BIM tools. This application will hence provide a set of powerful visualization and navigation tools that could be widely accessible to professionals and organisations, leading to the more inclusive adoption of BIM technologies.
2. **Easy Interaction**: This new way of interacting with BIM models that hosts a Conversational Agent will be able to process natural language, straightforward and intuitive commands, and execute complex operations in such a way that it will reduce the learning curve needed when dealing with traditional BIM applications. This feature is very important in relating the democratization of the BIM process among non-experts.
3. **Practical application**: The designed model fulfills the requirements of infrastructure projects, which, quite often, are very large and complex in nature. This feature of aligning visualization and handling models would address one important lacuna already in the variety of BIM software, with most of those tools being patently biased toward building projects. A practical orientation of this nature would ensure that the tool fits real needs to enhance efficiency and accuracy in managing infrastructure projects.
4. **Open Source**: By focusing on open-source projects, such research allows for more innovations and further collaborations within the AECO industry. As the application is open-source, it opens wide for any kind of contribution by worldwide developers and provides continuous improvement toward new challenges.

5. Impact on Education: An openly available IFC Viewer will also create a much more far-reaching impact than what is seen in education today. Future designers and contractors will then be able to learn and work with advanced BIM functionality without the barrier of cost. It would solve one very critical issue: it will close the gap between academic learning and practical applications.

Recommendations for further development and research based on the findings of and limitation found in the current study could be made as follows:

1. Performance Optimization: Work should be directed at increasing the performance of the application for very big and detailed models. The potential here is in the use of modern rendering techniques and performance-boosting technologies.
2. Development of Analytical Tools: Structural analysis and clash detection are some of the other features that shall add value to the application, making it an integral solution.
3. Better NLP means that natural language processing applied by the Conversation Agent can be improved, enabling this technology to be more intuitive and perform even better for complex queries or commands.
4. Offline Mode: This application would be more versatile when it could work offline, in instances when internet connectivity is low or unreliable.
5. Security Enhancements: Building up confidence among users in handling sensitive project information by implementing strong data security and privacy protocols.
6. Allowing the users to customize and increase extensibility by adding plugins or modules would make it even more flexible to respond to the needs of the user.
7. Interoperability with other tools: Interoperability would increase significantly with a wide range of BIM tools and platforms, improving data sharing and exchange, therefore expanding applicability and capabilities.

Bibliography

- Abioye, S. O., Oyedele, L. O., Akanbi, L., Ajayi, A., Delgado, J. M., Bilal, M., . . . Ahmed, A. (2021). Artificial intelligence in the construction industry: A review of present status, opportunities and future challenge. *Journal of Building Engineering*, 44, 44. doi:<https://doi.org/10.1016/j.jobe.2021.103299>
- Akinosho, T. D., Oyedele, L. O., Bilal, M., Ajayi, A. O., Delgado, M. D., Akinade, O. O., & Ahmed, A. A. (2020). Deep learning in the construction industry: A review of present status and future innovations. *Journal of Building Engineering*, 32. doi:<https://doi.org/10.1016/j.jobe.2020.101827>
- Bilal, M., Oyedele, L. O., Akinade, O. O., Ajayi, S. O., Alaka, H. A., Owolabi, H. A., . . . Bello, S. A. (2016). Big data architecture for construction waste analytics (CWA): A conceptual framework. *J. Building Eng.*, 6, 144–156. doi:<https://doi.org/10.1016/j.jobe.2016.03.002>
- Building a "ChatBot" for Scientific Research - Scientific Figure on ResearchGate. (n.d.). Retrieved September 24 , 2024, from https://www.researchgate.net/figure/Google-Dialogflow-Agent-architecture_fig1_326922722
- buildingSMART. (2024). Retrieved from <https://www.buildingsmart.org/about/openbim/openbim-definition/>
- buildingSMART. (2024). Retrieved September 23, 2024, from <https://technical.buildingsmart.org/standards/ifc/>
- buildingSMART. (2024). Retrieved September 23, 2024, from <https://www.buildingsmart.org/>
- Costin, A., Adibfar, A., Hu, H., & Chen, S. S. (2018). Building Information Modeling (BIM) for Transportation Infrastructure - Literature Review, Applications, Challenges, and Recommendations. *Automation in Construction*, 94, 257-281. doi:<https://doi.org/10.1016/j.autcon.2018.07.001>
- Darko, A., Chan, A. P., Adabre, M. A., Edwards, D. J., Hosseini, M. R., & Ameyaw, E. E. (2020, April). Artificial intelligence in the AEC industry: Scientometric analysis and visualization of research activities. *Automation in Construction*, 112. doi:<https://doi.org/10.1016/j.autcon.2020.103081>
- Eastman, C. M., Teicholz, P., Sacks, R., & Liston, K. (2011). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. John Wiley & Sons.
- Greyling, C. (2020, January 30). *How To Create A Chatbot With Google Dialogflow*. Retrieved from Medium: <https://cobusgreyling.medium.com/how-to-create-a-chatbot-with-google-dialogflow-60616c2b802f>
- Heikkila, R., Kolli, T., & Rauhala, T. (2022). Benefits of Open InfraBIM - Finland Experience. *Proceedings of the 39th International Symposium on Automation and Robotics in Construction* (pp. 253-260). International Association for Automation and Robotics in Construction (IAARC). doi:<https://doi.org/10.22260/ISARC2022/0036>
- K., S. (2014). *InfraBIM-Sanasto*. Retrieved May 26, 2020, from Eurostep Oy: https://buildingsmart.fi/wpcontent/uploads/2013/10/InfraBIM_Sanasto_0-7.pdf

- Kemppainen, A., Kolli, T., & Heikkilä, R. (2024). Development of Online Course for Open Infra Built Environment Information Model. *Proceedings of the 41st International Symposium on Automation and Robotics in Construction* (pp. 1025-1032). International Association for Automation and Robotics in Construction (IAARC). doi:<https://doi.org/10.22260/ISARC2024/0133>
- Kolli, T., & Heikkilä, R. (2020). Education of Open Infra BIM based Automation and Robotics. *Proceedings of the 37th International Symposium on Automation and Robotics in Construction (ISARC)* (pp. 757-764). International Association for Automation and Robotics in Construction (IAARC). doi:<https://doi.org/10.22260/ISARC2020/0105>
- Liu, X., Eshghi, A., Swietojanski, P., & Rieser, V. (2021). Benchmarking Natural Language Understanding Services for Building Conversational Agents. In E. Marchi, S. Siniscalchi, S. Cumani, V. Salerno, & H. Li (Eds.), *Increasing Naturalness and Flexibility in Spoken Dialogue Interaction* (Vol. 714, pp. 165-183). Singapore: Springer. doi:https://doi.org/10.1007/978-981-15-9323-9_15
- Mayor, A. (2018). Made Not Born. In *Gods and Robots: Myths, Machines, and Ancient Dreams of Technology* (pp. 1–6). Princeton University Press. doi:<https://doi.org/10.2307/j.ctvc779xn>
- McTear, M. (2021). *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots, Synthesis Lectures on Human Language Technologies*. Springer Cham. doi:<https://doi.org/10.1007/978-3-031-02176-3>
- Pauwels, P., Zhang, S., & Lee, Y.-C. (2017). Semantic web technologies in AEC industry: A literature overview. *Automation in Construction*, 73, 145-165. doi:<https://doi.org/10.1016/j.autcon.2016.10.003>
- RootsBIMLLC. (2024, August 5). *Top 10 Must-Have BIM Software for Infrastructural Design and Modeling*. Retrieved 09 11, 2024, from Medium: <https://medium.com/@rootsbim/top-10-must-have-bim-software-for-infrastructural-design-and-modeling-87d7aa45f2b3>
- Russis, L. d., & Roffarello, A. M. (2023). Conversational Agents: Human-AI Interaction. Politecnico di Torino.
- Saka, A. B., Oyedele, L. O., Akanbi, L. A., Ganiyu, S. A., Chan, D. W., & Bello, S. A. (2023). Conversational artificial intelligence in the AEC industry: A review of present status, challenges and opportunities. *Advanced Engineering Informatics*, 55. doi:<https://doi.org/10.1016/j.aei.2022.101869>
- WIKI.OSAARCH. (2024, September 13). Retrieved from WIKI.OSAARCH: https://wiki.osarch.org/index.php?title=IFC_-_Industry_Foundation_Classes/IFC_alignment
- Wu, H., & Dai, Q. (2021). Artificial intelligence accelerated by light. *Nature*, 589, 25–26. doi:<https://doi.org/10.1038/d41586-020-03572-y>