NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

ALEXANDRU ALIN CARAMIDA

Bachelor in Science and Computer Engineering

MODELLING OF 2D/3D COMPONENTS FOR OPTIMISING OPERATIONS IN INDUSTRIAL ENVIRONMENTS

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2022

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

MODELLING OF 2D/3D COMPONENTS FOR OPTIMISING OPERATIONS IN INDUSTRIAL ENVIRONMENTS

ALEXANDRU ALIN CARAMIDA

Bachelor in Science and Computer Engineering

Adviser: Nuno Tiago Marujo da Silva Santos Pereira
Director, WorldIT

Co-adviser: Pedro Manuel Corrêa Calvente Barahona
Full Professor, NOVA University Lisbon

Examination Committee

Chair: Teresa Isabel Lopes Romão
Associate Professor, FCT-NOVA

Rapporteurs: Fernando Pedro Reino da Silva Birra
Assistant Professor, FCT-NOVA

Nuno Tiago Marujo da Silva Santos Pereira
Director, WorldIT

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2022

Modelling of 2D/3D components for optimising operations in industrial environments

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.8.6) [1].

Acknowledgements

This dissertation would not have been possible without the help and support of the WorldIT family. I especially want to thank Nuno Tiago Pereira and Luis Saraiva for helping and guiding me when I had questions or faced a problem.

I want to thank my professor Pedro Barahona who together with Nuno Tiago Pereira made this dissertation possible, guided me and believed in me.

I want to thank my parents and my sister for constantly motivating me and being by my side all these years in university.

Thank you to all my friends and people who supported me and made this possible.

Abstract

Performing continuous evaluations and measurements of industrial components is extremely important to assess wear and maintain safety and quality levels. However much of this process is still done manually and without digital systems, which can be unproductive and more prone to errors. The adoption of a digital solution with 2D or 3D visuals of components and associated data can make regular inspections more efficient, improve data gathering and management, and warn inspectors about deviations of position or readings in new measurements. This dissertation proposes a system with graphic representations of the components and the ability to identify points of interest in them. Research is presented for technologies and software stack studied and used for developing this system. The architecture and implementation of this system are detailed along with examples of the current capabilities. The main focus was the ability to register measurements, both on-site and back office, and visualise them on an interactive map. The data can then be accessed by the client and his customers in a back office. Inspectors can also access the system on a mobile application, which allows them to register new measurements or view old data remotely.

Keywords: digitalization, 3D, 2D, inspections, industrial, measurements, components.

Resumo

A realização de inspeções e medições contínuas de componentes na indústria é extremamente importante para avaliar o desgaste e manter os níveis de segurança e qualidade. No entanto, grande parte deste processo ainda é feito manualmente e sem sistemas digitais, o que é menos produtivo e mais propício a erros. A adoção de uma solução digital que incorpore visualização 2D ou 3D dos componentes e dados associados pode tornar as inspeções mais eficientes, melhorar o registo e gestão de dados e alertar os inspetores sobre desvios de posição ou leituras em novas medições. Esta dissertação propõe um sistema com representação gráfica dos componentes com a capacidade de identificar pontos de interesse nos mesmos. Pesquisa é apresentada sobre as tecnologias e software estudados e utilizados para o desenvolvimento deste sistema. A arquitetura e implementação do sistema são detalhadas e uma demonstração com exemplos das capacidades atuais. O foco principal foi a possibilidade de registar medições, tanto no local e no back office, e visualizá-las num mapa interativo. O acesso aos dados pode ser feito através de um back office. Os inspetores também têm acesso ao sistema com uma aplicação móvel, o que lhes permite registrar remotamente novas medições.

Palavras-chave: digitalização, 3D, 2D, inspeções, industrial, medições, componentes.

Contents

List of Figures

List of Tables

Acronyms

2D Two Dimensional
3D Three Dimensional

API Application Programming Interface

CAD Computer Aided Design

GCS Geographic Coordinate System
GIS Geographic Information System
GPS Global Positioning System

JWT Json Web Token

LINQ Language Integrated Query

PCS Projected Coordinate System

SDK Software Development Kit
SQL Structured Query Language

VCS Vertical Coordinate System

1 Introduction

1.1 Description and Context

Inspections are an important task in every industry to maintain safety and quality levels and the heavy industry environment is no exception. This dissertation is being promoted by WorldIT and a client that performs inspections in industrial environments regularly. The client has a system to manage customers and orders and intends to improve the creation, access and interpretation of inspection data. The process of preparing, acquiring and interpreting data is still mainly done manually on paper and the available data is mostly unstructured which can produce inaccuracies or loss, leading to errors and important time waste. This dissertation aims to present a system that improves the data collection with the inclusion of visual elements to the inspections that can be integrated with the client's system. Since the inspectors work in onsite venues to perform the data collection the proposed solution includes a mobile module, optimised for performance and reliability. The dissertation focuses on the modelling and visualisation of environments with relatively big components, for example, gas tanks and boilers.

1.2 Motivation

Maintaining inspection data, unstructured or on paper records, can lead to unfeasible post-analysis and reporting. The client already has a system to manage customer info and order procedures for the inspections. The motivation was to improve the productivity of the preparation and execution of inspections. Additionally, it will be possible to improve the production of reports based on the collected inspection data and analyse them to predict possible flaws. A graphical representation of the industrial environment and its components can help interpret the data and aid visualisation and overall conclusions.

1.3 Solution

The graphical visualisation of the environments is an important part of achieving the client's objectives, so this dissertation focuses on this aspect. The placement of objects in the 3D scene is important to have an accurate positioning of components on the application map relative to the real world. This can be done by converting real-world coordinates to the application's space world or through graphic interaction, which was the solution adopted in this dissertation.

To help inspectors keep track of their inspection schedule, easy access to their assignments for the day is necessary. To facilitate the visualisation of inspection data inspectors should have the ability to create points of interest on the components that can be selected to consult or add inspection data. The consulting and addition of inspection data should be available both at the inspection location and back office. Users should have fast and easy access from different angles and perspectives to the 3D view of components and their inspected points, as well as the possibility of comparing new inspection data with past references and alerting for problems based on predefined levels of severity. Each inspection may include data that can improve its interpretation, for example, a picture, a description, measurements and a severity level. At last, there should be a user account system where each member of the team with proper access can view the data.

The functionalities mentioned in this section are only the proposed features of the system and can differ from a final build. The conclusions chapter of this dissertation will mention what was not implemented and propose future additions and updates to the system.

1.4 Report Structure

After the introductory chapter, the dissertation is structured as follows.
Chapter 2 - Related Work: This chapter presents the concepts of this work: Two Dimensional (2D) and 3D Graphics, Computer Aided Design (CAD) software, Game Engines, Georeferencing and the technologies studied for its development. It also gives some context for the project with a brief mention of what inspection software is already available.
Chapter 3 - Architecture: In the third chapter the architecture of the system is described. After the introduction of the system and the decisions made, the chapter proceeds to describe the different modules implemented and their purpose. At the end of this chapter, the hardware and software context are mentioned.
Chapter 4 - Implementation: The fourth chapter describes the implementation. In the previous chapter, it is explained how the different modules interact and relate to each other. This chapter explains the tools used for each module, the database structure and the process of creating the 3D models.
Chapter 5 - Results: In this chapter two user stories are presented to highlight the main features of the front end both on a browser and the mobile application.

Chapter 6 - Conclusions: The sixth chapter presents the conclusions based on the results and the evaluation of the previous chapters. It also mentions possible improvements and additional features that can be implemented.

2

Related Work

Since this dissertation focuses on the visualisation of environments and human interaction with the models, it was important to understand the options available to reach the objectives. This chapter compares the option of using 2D or 3D, which software is more suited for creating the component models and with which to build the graphics application. The possibility of using game engines for developing the application is presented with the analysis of two major game engines, Unity and Unreal Engine. Georeferencing techniques were researched for the accurate positioning of components but due to time constraints, it was not possible to implement. Currently, the industrial component placement is done manually by the developer when creating the scene on the game engine editor. This chapter ends with two examples of software already available in the market for scheduling inspections and reporting problems in an industrial environment.

2.1 Modelling and CAD Software

It is common for industrial environments to have detailed blueprints of buildings, piping and components, but historically on paper, and more recently in 2D CAD blueprints. Thus, software was studied to produce models of the components to include in the proposed solution. This section mentions various software options that were compared in terms of cost, community size, features and their integration with other technologies. A brief description of 2D and 3D is done to establish which path this project has followed.

2D environments use flat graphics to display length and height information on a flat surface without depth, usually photos or plan drawings. The user must understand the appearance of the final product, which may be difficult for people who do not have a technical background[2].

3D graphics make use of three-dimensional geometry, with Materials and Textures rendered on the surface of Objects to make them appear as solid environments. With 3D there is the possibility of using laser scanners to create the objects, saving time on the modelling process but it can be costly. A 3D model can contain various information about the points, lines, surfaces and entities of the product geometry which can make it

easier to show and explain products to consumers and workers[2].

An environment in 2D would limit the interaction and visualisation of the components when compared with 3D. The 3D visual can help both the inspector and the clients to have a better understanding of the environment while analysing inspection data. For this reason, it was decided that 3D would be the main focus for the graphics.

In this section 3D graphics toolkits, used for modelling objects, are described and analysed using a table of strengths and weaknesses for each one.

2.1.1 Meshroom

Meshroom is a free application that can create 3D models through the process of analysing photographic images. The amount of data, photos covering the object from all angles, varies with the object size and there is a need for good illumination in order to get photos with the best quality possible. The models generated by this application can be exported to modelling software to be edited, for example, cutting parts of the object that are not necessary.[3] With Meshroom we can have a starting point when modelling an object, which can save time and improve the fidelity of the model. For this reason, this tool was chosen to be used for testing, an example of its use and why it was chosen is later presented in the implementation chapter. Table 2.1 highlights the main strengths and weaknesses of Meshroom.

	Meshroom
Royalty fee	o - No royalties
Free Use	o - Free to use
Cross-Platform	x - Windows only
Community	x - Small community
Scan support	o - Can create objects using photos

Table 2.1: Meshroom Strengths and Weaknesses

2.1.2 Scaniverse

Scaniverse is an application for Apple devices that allows the capture, editing, and sharing of 3D content. It uses the LiDAR sensor available in recent iPhone and iPad devices to build 3D models with high fidelity and detail.[4] The models can be exported in various formats and easily imported to game engines mentioned later in this chapter. This application is appealing for being simple and fast to use. At the same time, it has the hardware limitation of requiring an iPhone or iPad with a LiDAR sensor and this sensor has a 5-meter range limit which could be a problem for creating larger objects. This application was chosen along Meshroom to make objects, an example of its use and why it was chosen is later presented in the implementation chapter. Table 2.2 highlights the main strengths and weaknesses of Scaniverse.

	Scaniverse
Royalty fee	o - No royalties
Free Use	o - Free to use
Cross-Platform	x - iOS only
Community	x - Small community
Scan support	o - Can create objects using LiDAR scan

Table 2.2: Scaniverse Strengths and Weaknesses

2.1.3 Blender

Blender is a free and open source 3D computer graphics toolkit used to create animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, virtual reality, and computer games. Importing a Blender model to a Game Engine can be as simple as drag and drop [5, 6].

Although hard to learn for beginners Blender has one of the biggest online communities with extensive documentation[7]. A big plus for Blender is being open source and completely free to use, which can make it an attractive choice for lower-budget projects. This application was chosen as an auxiliary tool for post-processing objects produced by Meshroom and Scaniverse, for example, to remove unwanted extra content from the resulting object. Table 2.3 highlights the main strengths and weaknesses of Blender.

	Blender
Royalty fee	o - No royalties
Free Use	o - Free to use
Cross-Platform	o - Windows, Mac, Linux
Community	o - Large amount of material and videos online
Scan support	x - Object made by hand

Table 2.3: Blender Strengths and Weaknesses

2.1.4 SolidWorks

SolidWorks is a solid modelling CAD computer program that runs primarily on Microsoft Windows [8]. Just like Blender, its models can be converted to formats supported by game engines, which can help produce components with high graphical fidelity and easily import them to other tools. There is also the possibility to import scanned objects to SolidWorks using scanners and plugins from other companies, for example, Artec3D, which will be analysed next.[9].

It can take a while to learn for beginners but has a large number of online materials for learning, but the cost associated with it can be a limitation for low-budget projects. Table 2.4 highlights the main strengths and weaknesses of SolidWorks.

	Solidworks
Royalty fee	o - No royalties
Free Use	x - Software (pricing on quote request)
Cross-Platform	o - Windows, Mac
Community	o - Large amount of material and videos online
Scan support	o - Can use scanned data to create objects

Table 2.4: SolidWorks Strengths and Weaknesses

2.1.5 Artec3D

Artec is a private enterprise that offers modelling software, the latest version is Artec Studio 16, which can be used with scanners for modelling real-world objects. Additionally, Artec offers a Software Development Kit (SDK) for integration of their scanners with other software.

The Artec Scanning SDK enables the development of a scanning app to control Artec3D scanners and process the captured data, add support for the scanners to software, or create C++ plugins for other industry-recognised software [10].

In terms of model accuracy and measurements, Artec scanners may have the best result but the scanners come at a high price. The option to use the SDK to develop a scanning app can be too complex and time-consuming. The possibility of using Artec Studio 16 seems more viable but comes at a high monetary cost. The scanner itself needs to be a model with support for large-scale objects which Artec advertises as "Artec Ray"[11]. This tool can be considered a more sophisticated Scaniverse since both use scanning to produce models, but the proprietary software and hardware with very high prices make it impossible to consider for this dissertation. The Table 2.5 highlights the main strengths and weaknesses of Artec3D.

	Artec Studio 16
Royalty fee	o - No royalties
Free Use	x - Software (€2000 or €800/year), scanner (€56000)
Cross-Platform	x - Only Windows
Community	x - Most material available is from Artec
Scan support	o - Can use Artec scanners to model objects

Table 2.5: Artec3D Strengths and Weaknesses

2.2 Game Engines

Game engines are technologies that allow designers to efficiently code and plan out a graphic environment without having to start from scratch. They provide a software development environment that assists designers with asset generation and placement, whether 2D or 3D based. This section compares two of these engines, Unity and Unreal

Engine, with the intent of studying their strengths and weaknesses and deciding which to use.

2.2.1 Unreal Engine

Unreal Engine is a game engine that was initially exhibited in the 1998 first-person shooter game Unreal. Originally designed for first-person shooters, it has since been adopted by other industries, most notably the film and television industries, and is now utilised in a number of 3D gaming genres[12].

The Unreal Engine is written in C++ and is extremely portable, supporting a wide range of desktop, mobile, console, and virtual reality platforms[12]. When building the game code itself, the engine uses a combination of Blueprint (a proprietary language unique to Epic products) and C++[13]. It is popular in computer and console games with high graphical capabilities, and it is utilised in a variety of games as well as filmmaking and other business applications.

Epic uses a royalty mechanism to allow it to be used in commercial products, charging creators 5% of sales revenue after their first US$1 Million in income[14]. Unreal was not used because of no past experience using Blueprints and having a harder learning curve than Unity. Table 2.6 highlights the main strengths and weaknesses of Unreal.

	Unreal Engine
Royalty free	o - 5% royalty but only if the revenues exceed US$1 Million
Free Use	o - The editor is free to use
Cross-Platform	x - Only Windows
Community	o - Large amount of online material and store with over 16K assets
Past experience	x - No past experience with the engine
Updates	o - Regular updates
Access to source code	o - All user groups can have access to source code
Learning Curve	x - Takes months to learn basics for beginners. Steeper than Unity
Mobile support	o - Can develop apps for Android and iOS
Model integration	o - It uses .fbx, which is a standard file type
Proprietary Language	x - Uses C++ with Blueprint. Blueprint is unique to Unreal Engine
Rendering	x - Geared towards high-end devices can get heavy for mobile

Table 2.6: Unreal Strengths and Weaknesses

2.2.2 Unity

Unity is a cross-platform game engine developed by Unity Technologies, the engine supports a wide range of platforms, including computer, mobile, console, and virtual reality. It is especially popular for iOS and Android game development, with titles like Cuphead utilising it[15].

Unity allows users to create 2D and 3D games and applications, with the engine having a core scripting Application Programming Interface (API) written in C# for both

the Unity editor and the applications themselves. Importing models to the editor can be as simple as drag-and-drop[15]. The fact that it is simpler to use when compared with other engines, is free for personal and student use and has an easier learning curve made it one of the most popular game engines.

The engine also became a major force outside the gaming industry namely the automakers, brands like Volkswagen, Audi, BMW, Volvo, Lexus, Lincoln, Toyota and others use the engine[16]. The main attraction of the engine is the ability to create very high-quality visualisations of complex models in real-time. Designers can easily make changes to the models in CAD, bring the model into Unity, render it instantly with realistic materials, and then explore it with great interaction and immersion on their devices.

In terms of price, Unity charges for using its editor commercially if the project budget or revenue is over US$100K but does not charge royalties[14, 17].

Although being a paid software in a professional setting, Unity is free to use for education purposes. The fact of having some experience with the user interface made it the choice for developing the application. Table 2.7 highlights the main strengths and weaknesses of Unity.

	Unity
Royalty fee	o - No royalties
Free Use	x - Free personal use, commercial (from US$ 399/year)
Cross-Platform	o - Windows, Mac, Linux
Community	o - Large amount of online material and store with over 65k assets
Past experience	o - Basic experience with the engine
Updates	o - Regular updates
Access to source code	x - Access granted on request and for paying members only
Learning Curve	x - Can take months to learn basics for beginners
Mobile support	o - Can develop apps for Android
Model import	o - It uses standard file types, drag and drop Blender objects
Proprietary Language	o - Unity uses C#, a well know free language
Performance	x - Optimisation may be needed for large and complex projects

Table 2.7: Unity Strengths and Weaknesses

2.3 Georeferencing

Georeferencing is the technique of giving positions to geographical objects within a geographic frame of reference. The visualisation of the environments is a focal feature of this work and with it the positioning of the components and equipment in the application space. Georeferencing is crucial to geospatial technology in general and geographic information systems to position the components with accurate coordinates on a spatial reference. Georeferencing processes may be divided into two types based on the spatial resolution in use: metric georeferencing and indirect georeferencing.[18] Metric georeferencing is a coordinate-based method of georeferencing, a collection of values in a

coordinate system may be used to specify any point on the environment surface. Indirect georeferencing approaches use attribute data to get metrically georeferenced locations from existing metrically georeferenced Geographic Information System (GIS) datasets. GIS databases are collections of geographic features referred to by coordinates and rely on metric georeferencing[18].

The spatial reference is used to describe the position of features in the real world. It includes a coordinate system for x, y, and z as well as a tolerance for said values and the meridian. In a coordinate system, the x and y coordinates are georeferenced with a geographic or projected coordinate system. A Geographic Coordinate System (GCS) is defined by a datum, an angular unit of measure (usually degrees), and a prime meridian. A Projected Coordinate System (PCS) consists of a linear unit of measure (usually meters or feet)[19].

It can have a Vertical Coordinate System (VCS) as an optional property, the z-values, which are commonly used to represent elevation or depth values. According to Esri, the most important parts of a vertical coordinate system are its unit of measure (for example, international feet or meters) and whether the z-values represent heights (elevations) or depths[20].

A spatial reference also includes tolerance values for the coordinates, all have associated tolerance values that reflect the accuracy of the coordinate data. The tolerance value is the minimum distance between coordinates and if one coordinate is within the tolerance value of another, they are interpreted as being at the same location. Esri recommends in most cases a tolerance value that is 10 times the default resolution value[19]. The resolution defines the minimum distance, in map units, that separates unique x-values and unique y-values in the feature coordinates. For example, if a spatial reference has an x, y resolution of 0.01, then x-coordinates 1.22 and 1.23 can be stored as separate coordinate values, but x-coordinates 1.222 and 1.223 are both stored as 1.22[19].

The georeferencing technique was researched with the focus of developing a module to position the components with some degree of accuracy on the coordinate system of the application. With inspections using various types of equipment for measurements, georeferencing coordinates could have a big impact in the future with importing data correctly to the visual environment. But due to time constraints, the focus was the modelling of objects and the creation of the application. Georeferencing was thus put on hold as there was no time to develop it.

2.4 Inspection Software Examples

Regarding industrial inspections, there is multiple software available, including cloud-based, that rely heavily on pictures and written reports, on many of the searched public examples they do not have a visual representation of the environment and do not appear adequate for large-sized components that need sensitive and accurate measurements.

This subsection presents two examples of inspection applications that offer cloud-based storage for inspection data, maintenance schedules and reports.

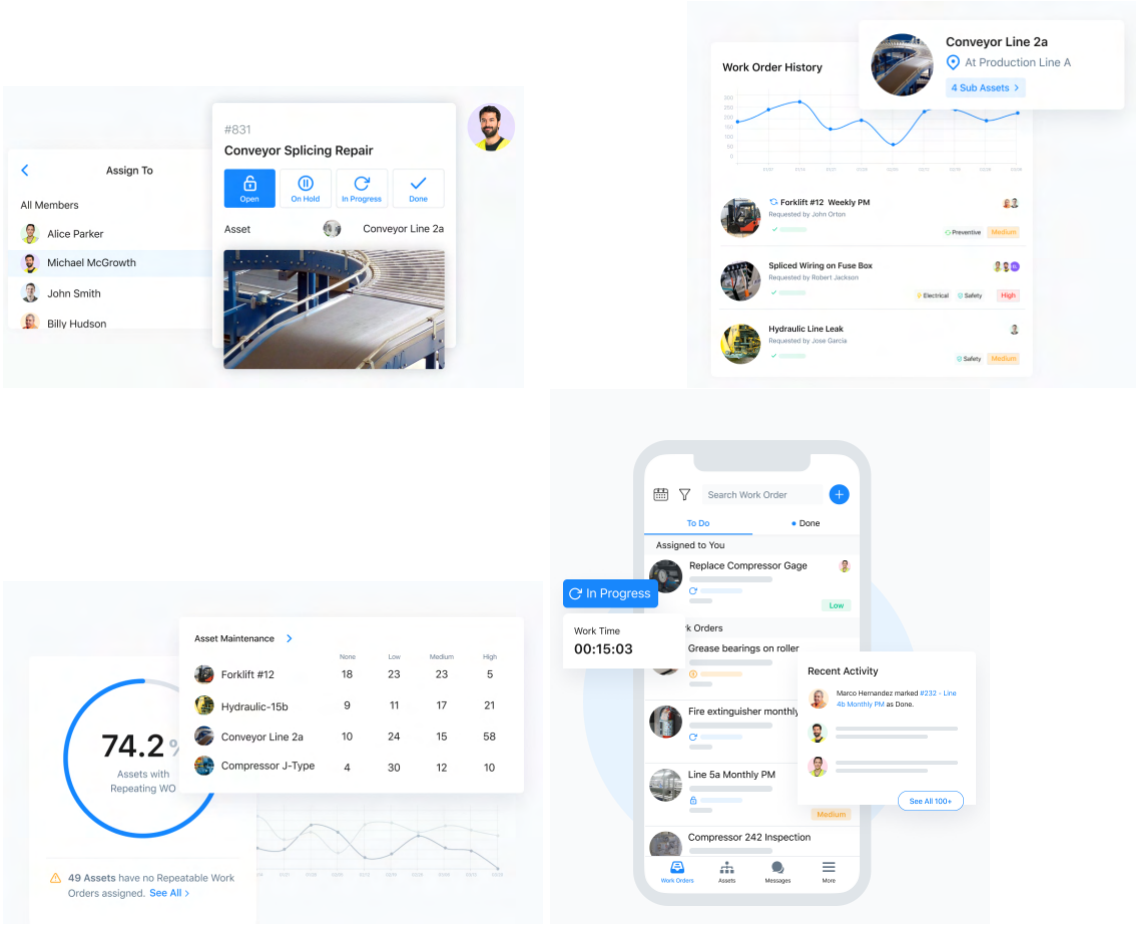

Figure 2.1: Some interfaces of application MaintainX.

MaintainX(figure 2.1), states its solution is oriented toward facilitating manufacturing operations and reducing down time[21]. This application offers the ability to assign work orders to technicians to fulfil maintenance requests for equipment, keep a complete maintenance log, digitise all data entry and generate reports from it and the possibility to track work requests across assets and locations[21].

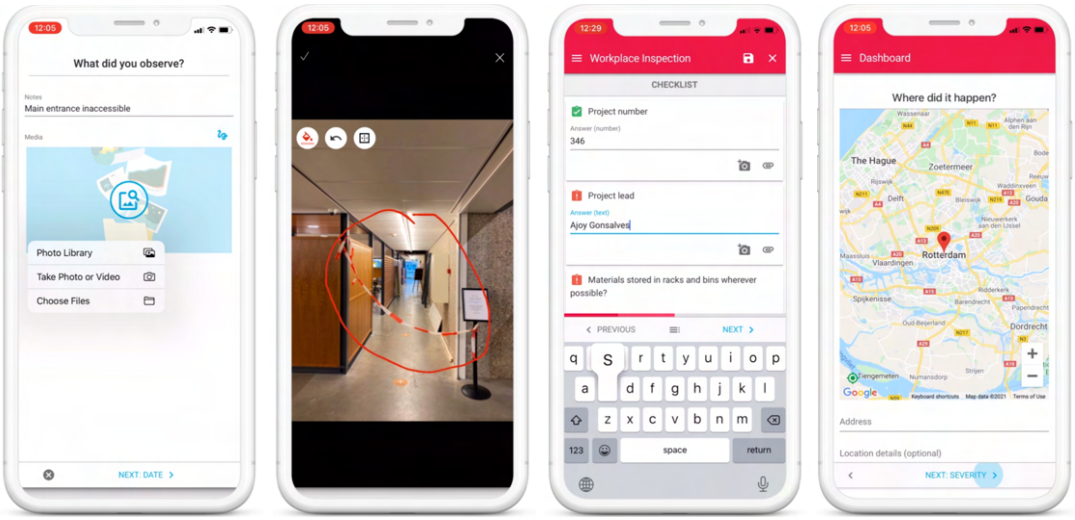

Figure 2.2: Some interfaces of application Capptions.

The second example is Capptions (figure 2.2) and it is similar to MaintainX but advertised more towards workplace safety. Capptions state that they are a platform which connects the workforce by providing a single space to store essential safety-related data, key environment, health and safety performance metrics, information on tasks, and more[22]. They offer a no coding or prior experience needed approach to creating forms for the reports, the possibility of attaching photos and Global Positioning System (GPS) location to them and the ability to analyse data to find trends using graphs. They offer a wider range of features that can be consulted on their page[22], but Capptions seems more in line with the system proposed in this dissertation and a good inspiration for similar features.

Both these examples do not support 3D models of the components, which is the main feature this project proposes. They also do not seem to support specific measurements of components and are more focused on ground worker reports and resolution assignments by managers.

Hence a simple inspection component was developed in this work, that may be integrated with the client's system already in use.

3

Architecture

Initially, the plan was to implement the application with support for both 2D and 3D but because of time scope limitations and after some 2D tests, it was decided to focus on 3D. Considering the objectives established it was decided that the 3D view of components and registration of inspection data onsite should be the first to be developed. When considering the objective of having a 3D view of the environment for the inspectors we have to consider the visit to locations to perform the inspections. For this reason, it was decided to create a mobile application with the intent of providing a quick and easy visualisation of the scene to inspectors wherever they are. Some of the features were made with flexibility in mind so that any necessary future improvements can be made without architecture disruption. The system was divided into five modules: Services, Web App, Database, Mobile App and WebGL (see fig. 3.1). The Web App and the Services were built with ASP.NET Core using Visual Studio 2019. The 3D for the web was built with Unity using WebGL API and integrated into the Web App using an inline frame. The Mobile App works with Android devices and was developed with the game engine Unity using the Visual Studio 2019 editor and Android 10 as the minimum operating system. For the Database, SQLite was chosen for being lightweight, easy to integrate and free to use. SQLite being a SQL database engine in the future can be replaced by a more sophisticated and scalable SQL engine. The modelling of the 3D objects is currently done manually with the assistance of the tools referenced in chapter two.

The following section explains the architecture of this system and each of the distinct modules that compose it. At the end of the chapter, the context and the current hardware and software limitations of this system are mentioned.

3.1 Architecture

The architecture has been defined in three layers, the Backend, the Browser(web front end) and the Mobile. The Backend layer is composed of the Database and the Services which are responsible for the communication of the Database with other modules. The Web App and WebGL modules are also part of this layer, with WebGL rendering the 3D

view on the front end and the Web App is the coding of the web pages. The Browser layer represents the internet browsers used for the front end. Finally, the Mobile layer consists of the user front end on mobile devices (mainly used by the on-site inspectors), and is composed of the Mobile App module alone.

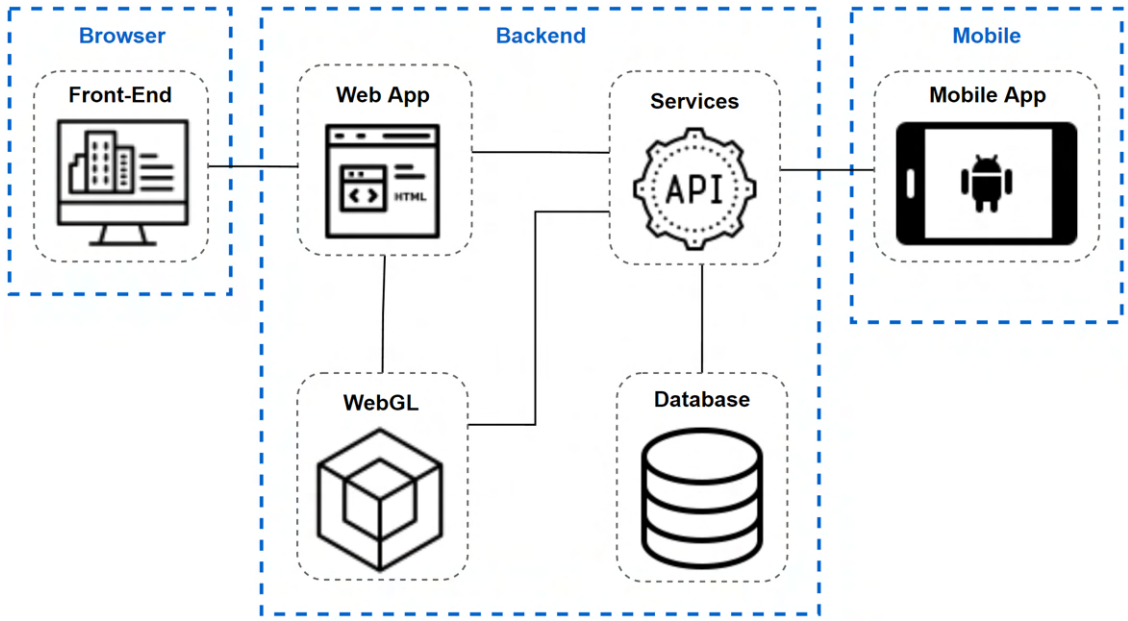

Figure 3.1: Architecture layers and their modules.

In figure 3.1 the diagram shows how the modules from these layers interact with each other. The Services module is used by the Web App module, the WebGL module and the Mobile App module to access the database. The WebGL module and the Web App module interact with each other to provide access to the 3D environment and the Front-End is the internet browser view of the Web App. The system needs these five modules to achieve the objectives with each one having a specific purpose.

- **Database:** The Database is where the data of the system is stored, it is Structured Query Language (SQL) based and was made with the SQLite engine, this module will be further explained in the implementation chapter.
- **Services:** The Services module represents the implementation of a RESTful service that makes the connection between the Database and the other modules, it decouples the data storage and retrieval by the other or any future modules.
- **WebGL:** The WebGL module permits viewing the 3D environments on the Front-End(browser) module, it is made with Unity using the WebGL API. WebGL is a API for rendering interactive 2D and 3D graphics within any compatible web browser.
- **Web App:** This module is the code for the web pages accessible to authenticated users through a web browser, allowing the management and consulting of data.

- **Front-End:** This simply represents the browser view of the Web App module using an internet browser, at the end of this chapter the browsers used are mentioned.
- **Mobile App:** The Mobile App module represents the Android application used by an inspector to visualise the environment and register inspection data.

3.1.1 Services

This module consists of an implementation of RESTFUL web services that allow client entities like browsers, devices or tablets to reach the data. While implementing this module care was taken for future updates or modifications. If there is a need to substitute the database engine it would only need to replace the connection string used to connect to the database and import the necessary dependencies of the engine. Another important consideration was to prepare the API for authentication and authorisation, which was achieved using roles for the users and a Json Web Token (JWT) for authentication. In general, the API was made with the need to expand in mind, if an operation that is not implemented is needed it can be added without major problems. The endpoints that can result in big lists are also prepared for pagination to reduce the size of the data returned.

3.1.2 WebGL

There was a need to have a 3D visualisation of the locations in the browser front end to make it more easily accessible to both clients and operatives, including inspection planners. To achieve this the 3D part of the project was divided into two modules, one for the web and another for mobile. This module is relative to the web, it shows the currently selected venue of a location in 3D. The user can explore the venue using the keyboard and mouse and select objects to consult their marked points of interest or create new ones, which are represented by small coloured dots, and in turn, the inspections associated with each point.

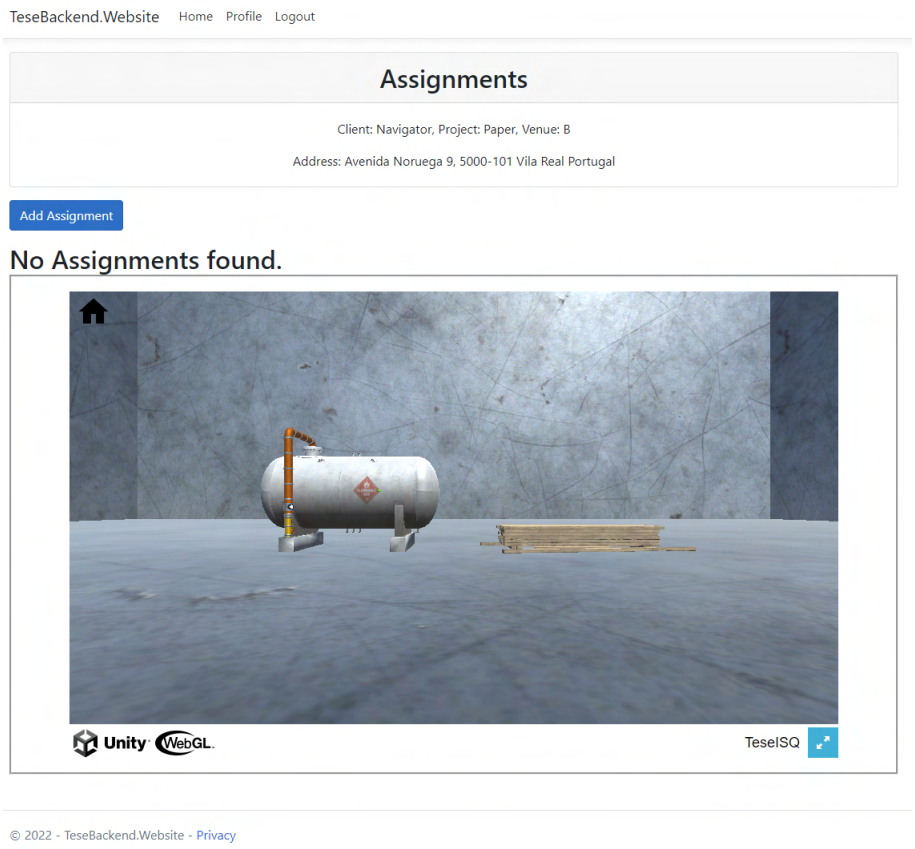

Figure 3.2: The 3D frame in the Assignments page.

The integration of this module and the Web App module is made using an inline frame where the WebGL API renders the 3D environment, an example of this can be seen in figure 3.2. The information on which venue to render and the current user token is passed from the Web App module to the WebGL module using the browser's local storage. With the id of the venue, the respective 3D scene is loaded for the user, the token is necessary for the web requests authorisation and authentication. If the venue does not have yet a 3D scene an error message is displayed.

3.1.3 Web App

This module is the web page for the web front end that users can access using an internet browser. The user is greeted with the login page with the credentials being created by an administrator and passed on to the users since this is supposed to be an internal system. The users have roles and that determines the type of operations they can perform on the website.

- **Admin** - The administrator that has full control in terms of creating, editing and deleting data. This role represents a manager or planner that creates assignments

for the on-site inspectors, adds new clients to the system and creates new user credentials. An admin can also perform inspections.

- **Technician** - Represents the users that go to locations to perform inspections. Besides consulting past data they can create new inspections, edit them and create points of interest on components.
- **Client** - This role is given to client individuals that get their locations inspected. Using the solution these users can consult the company and inspection data and the 3D view of the locations.

3.1.4 Mobile App

This module has many similarities with the WebGL module but was made with the intent of being an independent application for the inspector. Although being different projects, the material can be shared between this module and the WebGL module to reduce development time when adding new environments. For example, a 3D scene of a venue can be constructed on the WebGL module and then copied to the Mobile App module with very few modifications needed. This way the time it takes to create a new venue in the system will be reduced. The web request implementation is also the same on both projects since the data is treated equally on both modules. The main differences are the user controls and the user interface.

The user interface is more robust on this module, there is a need to login, it lists the user assignments, adds the ability to upload a photo to each inspection and the 3D scene has a menu to manage the assignment.

Figure 3.3: Mobile App 3D scene with sidebar menu opened.

An example of the menu can be seen in Figure 3.3 with the sidebar menu showing the assignment status. To navigate on the 3D scene the user uses his fingers on the screen simulating joysticks, the left side of the screen is movement and the right side is rotation. The 3D scenes on this module are only accessible using the Schedule pages of

the application. For a user to access a venue on the Mobile App he needs to have had at least one assignment for that venue.

3.2 Context

In this section, some current software requirements and system limitations are mentioned. This system is aimed at internal use only, with the inspectors and managers being the only ones with access to the Mobile App and the client only having access to inspection data through a web browser. The Mobile App requires an Android device, preferably a tablet, with API 29 or greater, this was mainly because the device available for testing and being a fairly popular version with 73.32% of devices using Android 10 or newer[23]. A caching system was developed for the 3D applications but is aimed at preventing repeated web requests, which means the application can only be used in environments with an internet connection. For the web version the web browsers Mozilla Firefox, Microsoft Edge and Google Chrome were tested and the web pages displayed as intended on all three browsers.

4

Implementation

This chapter presents the tools and frameworks used in this thesis implementation of the system's modules and their functionalities. It also explains the process of modelling an object and the database structure.

4.1 Component Modelling

The first step was to define the creation method of the 3D models that will be imported to the application. Based on the study conducted in chapter two, it was decided to proceed with Meshroom and Scaniverse for the model creation and edit them when needed with Blender. There is also the option of using generic models available on the Unity Store if model fidelity is not necessary. Initially, the process was done by taking photos of the object from all angles in an illuminated environment and using Meshroom to process them into a 3D model. This method has high hardware requirements, namely a powerful graphics card with support for the CUDA API. The processing time increases with the number of pictures which can make the modelling of big objects take a considerable time. Besides this, it requires good illumination of the environment and it has problems rendering reflective surfaces like windows.

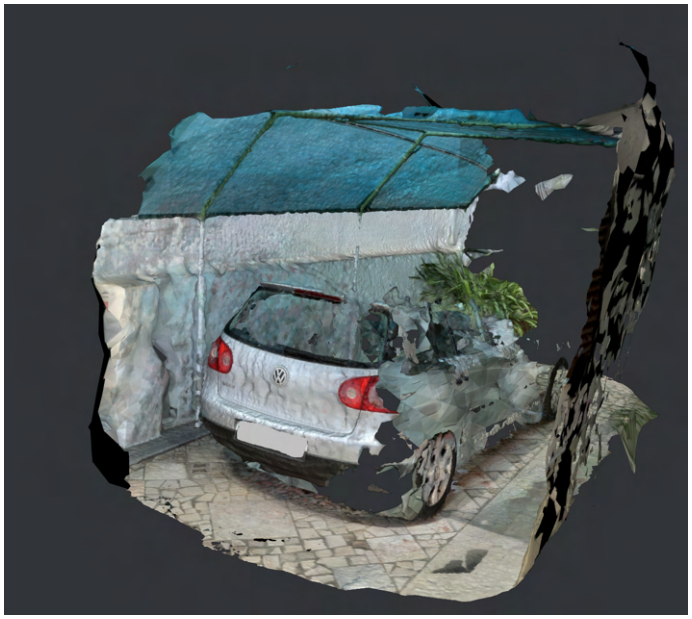

Figure 4.1: Car model processed by Meshroom using 55 pictures.

A bad set of data can result in unusable models as figure 4.1 shows, and the processing time of a big object will also be higher as the number of pictures increases. The example of figure 4.1 took just over 21 minutes to process with 55 pictures and the result is unusable. Having the object in an open space with good illumination can give better results but that is not always possible to achieve in industrial environments. Faced with these challenges, other alternatives such as Scaniverse and the generic models were approached.

The mobile application Scaniverse uses the LiDAR camera found on newer iPhone and iPad devices to create a 3D model of a scanned area. The process is fairly simple: first, the application on a supported device is launched, then one clicks the record button and moves around the object until all red lines disappear.

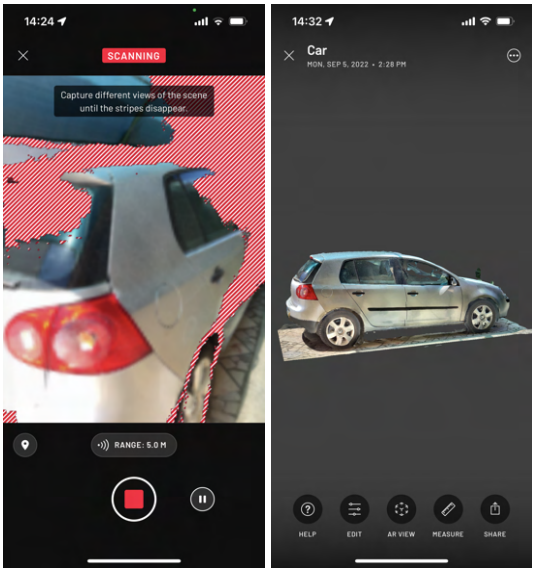

Figure 4.2: Scanning a car with Scaniverse and the result.

In figure 4.2 we can see the scanning process on the left image and the resulting model on the right image. This process is faster than Meshroom and the models can be edited and cut directly on the application. The processing time of the car model in figure 4.2 took under one minute. To export the models we simply click the share button and select the format, Scaniverse can export multiple formats including those supported by Unity. The process has some limitations, the LiDAR camera of the device can only reach up to 5 meters, it also requires good illumination of the environment and has problems rendering reflective surfaces [24].

The ability to edit models on the application itself, the option to export them in various formats and the fact that a compatible device was available made it a strong candidate. All this plus the no cost to use, the relatively good results, simpler use and faster processing time when compared to Meshroom made Scaniverse the choice for the modelling process.

There is still another option, to use a generic object already made by someone, with the community size of Unity many 3D models can be found online. The problem would be the details and fidelity since the object's appearance and the model would be different. Later in the results chapter, an example of both models made by the community and models made with Scaniverse will be presented.

4.2 Backend layer

This layer establishes a connection between the database and the modules from the other layers. For those reasons, it should be reliable, easy to maintain and scalable, and so it was decided to use the Azure App Service for deployment. With the Azure App Service,

an app can be deployed for any platform or device on a scalable and reliable cloud infrastructure[25]. The main reasons to use Azure in this project are the fact that WorldIT has worked with it in the past, which makes it easier to maintain or modify, and also if support was needed they have the experience and knowledge to provide it. Microsoft offers a student plan with credit to use on their cloud platform and that was also a factor.

4.2.1 Database

Based on feedback from the instructors and past experience using SQL it was decided that the database will be built using this standard. With the project being in a proof of concept stage it was decided to use a SQL engine that is light with a small footprint and allows the data to be easily transferred to other engines at a later date. The engine chosen was SQLite for being a compact library but having most of the SQL features, furthermore, the engine reads and writes directly to ordinary disk files with a format that is cross-platform, you can freely copy a database between systems[26].

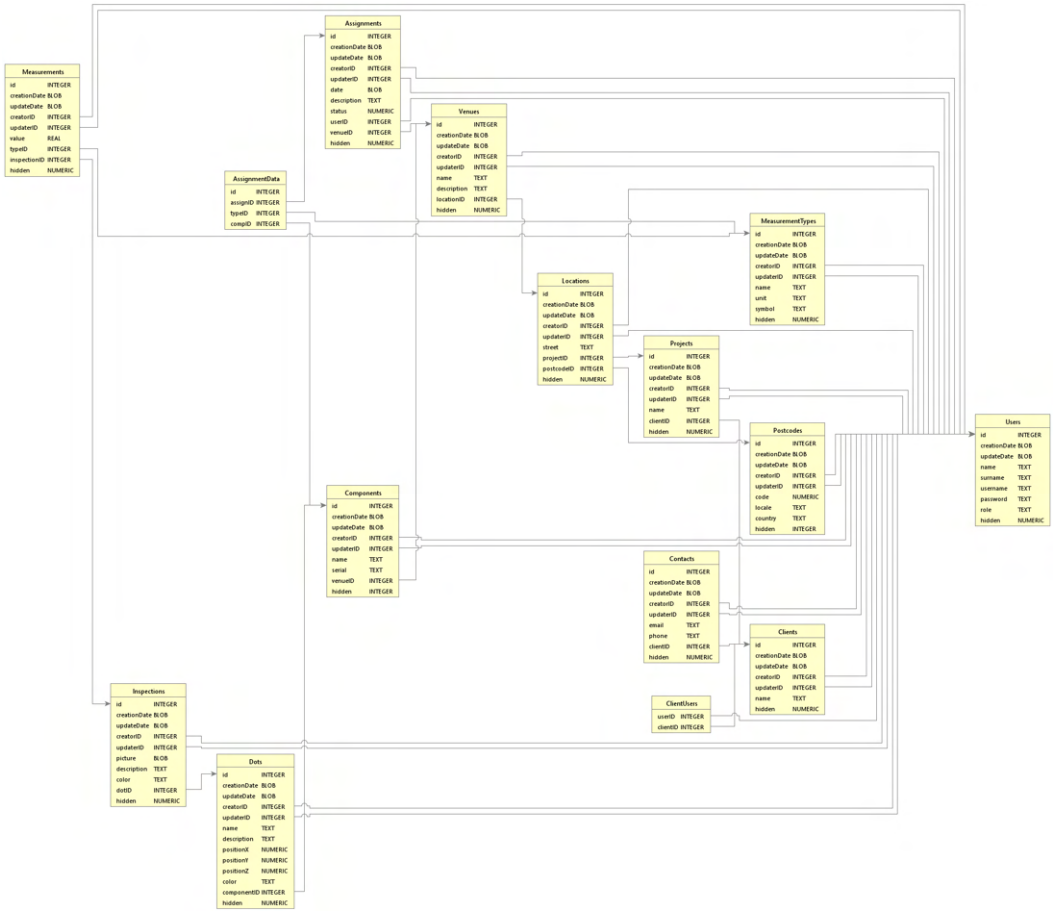

Figure 4.3: Database Entity Relationship Diagram.

In figure 4.3 we can observe the Entity Relationship diagram to help us understand how the data is structured. In Annex I the tables are shown in detail.

At first glance, we can see that almost all other tables have a relationship with the Users table (on the right). This structure is a design choice since we want each element of the table to have a record of who was the creator or who modified the data. With the intent of good practice and safety, all tables have a value for the date of creation and date of modification alongside the creator and the updater. The hidden value is present in almost all tables as well, its purpose is to deactivate an element that no longer is in use without deleting it since that would raise dependency problems. The id for all tables is an integer automatically incremented by the SQLite engine. This way there will not be problems with complex identifiers. The database file was built using DB Browser for SQLite, which is an open-source tool to visualise, create, design, and edit database files compatible with SQLite[27].

The structure was originally planned to have some influence from the client as they already had a client list, and the types of inspections and they could give insight on the general structure of an inspection system. Since the feedback was not received the structure was built with limited knowledge of an industrial environment and so this database is a proof of concept.

Starting with a table for the **Users** of the system, it contains the standard information for a user, the name, surname, username and password. The password is encrypted with a simple string cipher in order to prevent the display of the password on the database. It also contains the role value which will be used to determine the authorisation privileges for the user, at this moment the roles are defined on the front end of the system.

The **Clients** table has the name of the company, which should be unique, and keeps relevant relations with other tables, the **Contacts** table which stores the contact information of clients, and the **ClientUsers** which is used to identify the users that represent a client. The **ClientUsers** table uses the User id as a key because a user can only belong to one client but a client can have multiple users.

Next, the **Projects** table has a name and the id of the client and was defined to organise and affiliate the Locations with a client.

The **Locations** have the street and the Postcode id in order for the inspector to have an address destination when supposed to conduct an inspection. The postcode is stored in the **Postcodes** table along with locale and country with the intent to prevent value repetition since there may be locations with the same postcode. A location can be divided into more than one division or venue and so the **Venues** table represents just that.

The Venues table has a name and description in order to better understand the layout and content of a venue. The venues are the place of the components that will be inspected and that is why there is a relation between the **Components** table and the Venues one.

The components have a name and a serial, which is unique to help identify them. They can have points of interest associated to better locate inspection details.

The **Dots** table stores data of points of interest, the dots are created by users and so their position needs to be stored in order to plot them in the right position when the application is initialised. This is done with the positionX, positionY and positionZ values.

The dots also have a name and description to better identify them, the description can be edited by users with technician or admin roles. Lastly, the dots have a colour value which represents the severity of the latest created or edited inspection. The severity colours will be better explained in the WebGL module subsection.

The **Inspections** table has a description to better understand what was done and the results of the inspection, a picture to store a photo of the Dot inspected and the colour for the severity. I would like to clarify that the inspections represent the inspection made to the Dots and not the inspection as a whole.

The **Measurements** table stores values for the inspection and their type, this was made with the intent that an inspection can have more than a measurement. Although this was the initial plan due to time constraints the application user interface will only register and show one measurement per inspection. The available measurement types are stored in the **MeasurementTypes** table, with the name, unit and symbol of the unit, an example would be Temperature, Kelvin, K.

Finally, the **Assignments** table stores assignments created by an administrator for a technician. These assignments hold data on which venue and consequent location should the technician go to perform an inspection. Furthermore, it has the date when to perform the inspection and a description to help describe the situation. In order to make an assignment description more detailed, the **AssignmentData** table was created, which stores the components along with the type of inspection to perform for each assignment.

4.2.2 Services

This module was built with the intent to act as a bridge between the database and the other modules. It is a RESTful service composed of ASP.NET Core controllers, the controllers support creating, reading, updating, and deleting operations.

ASP.NET is an open source web framework, created by Microsoft, for building web apps and services with .NET[28]. ASP.NET Core consists of modular components with minimal overhead, ideal to build flexible solutions and with support for multiple platforms, on Windows, Mac and Linux[29]. The framework version used on this project was .Net Core 3.1. Using this framework was a fast choice because WorldIT has experience with .NET, the language being C#, which is the same as Unity, and both have integration with the Visual Studio 2019 editor.

Pagination was implemented on methods that can return big lists, the methods receive as headers the page and page size. In order to prevent a call with too big a page size, the numbers are validated with the PaginationFilter class for a certain interval, if the size is bigger than the max allowed it defaults to it.

For a simpler modification or future update of the database .Net Language Integrated Query (LINQ) was used. This also contributes to an easier transition to a different SQL engine. The database context is added as a service to the container of the ASP.NET application, this way the connection is only declared in one place and easily modifiable.

The context is implemented using Microsoft's Entity Framework Core, which allows data access to be performed using a model. A model is made up of entity classes and a context object that represents a session with the database, this context object allows querying and saving data[30].

One other consideration was authorisation and authentication to prevent the access of data by unwanted parties. The authentication is achieved using a token created and returned when the user logs in to the system, the user logins with their credentials and if valid a token is generated and returned. The authorisation is done on the endpoint level by declaring which roles have the authorisation to access the said endpoint. The token is always passed on when making a web request and it has the role of the user in it, this way the system can determine if he has authorisation for that operation. For example, the creation of new users can only be made by a user with the role of "Admin". The JWT was chosen for the token implementation for its simplicity, easy integration, and being a compact and self-contained way for securely transmitting information between parties as a JSON object [31].

4.2.3 Web App

With this module, the objective was to have a back office with a browser front end where managers can create, edit and consult data but also clients can view the inspections performed with a 3D view of the locations. Initially, this module was not planned in its entirety since the client already had a system to manage its data and the type of inspections. During this dissertation, there was no further information from the client and for this reason, it was decided to implement a back office for the project.

Since the Services module was built with ASP.Net the front end was decided to be as well, this facilitates the implementation and deploying since both modules can be on the same deployment. The Microsoft course [28] was the main guide on how to build a core website.

The pages themselves were built using the Blazor framework, which is part of the ASP.Net Core framework and Bootstrap. Blazor allows the creation of client-side web user interfaces with .NET writing code in C# instead of JavaScript and benefiting from .Net performance, reliability, and security [32]. Bootstrap is a Framework for developing websites, a popular CSS language used to style an HTML document. In this module, Bootstrap 4 was used for the visual of the pages since it comes already bundled with the core website template from Visual Studio 2019.

The design of the website was made with simplicity in mind, since time was a concern only the essentials were implemented. With a minimalist design on all pages, the website welcomes users with a login page. On login, a token is created and stored on local storage and the user is redirected to a landing page, which varies on the user role. For an account with "Admin"or "Technician"roles the home page displays the client list but for an account with the "Client"role displays projects of the Client to which the user is associated. The

token is also verified on every page and redirects the user to the login page if it has expired, by default the token is valid for 30 days after its creation.

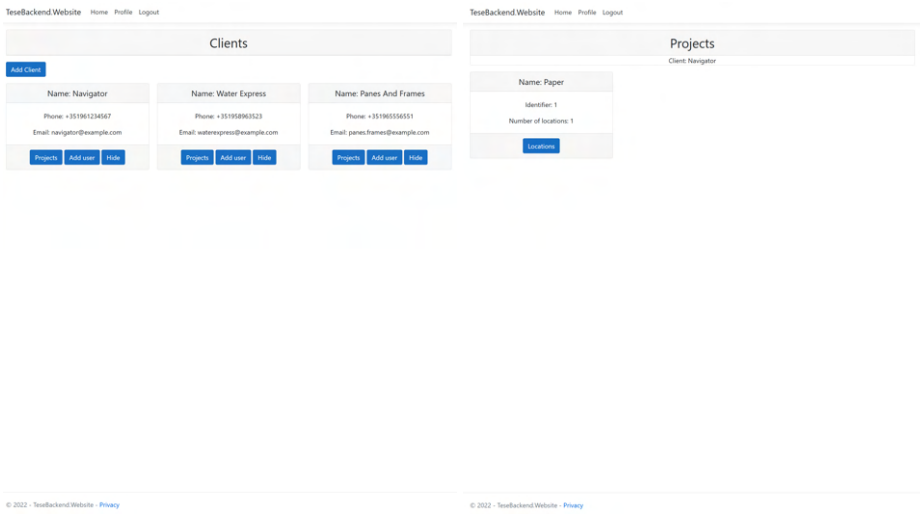

Figure 4.4: Admin home page on left and Client home page right.

In figure 4.4 we can see the different home pages resulting from the code described previously. Across all pages of the website, there are verifications for the roles and that affects the page design and functionalities available to the user. One example of this would be the ability to create new data, for now, a user with the "Admin"role is the only one able to see buttons for that task.

For creating forms there was a need to verify when the user submits the forms if the data is valid, for example, while creating a Client the name, email and phone number can not be empty, the Client name must be unique, and the phone and email must be of a valid format. In order to assist with these verifications, Data Annotations were used to define a certain field of a Model as required. If the user tries to submit a form with a required field empty a message will be displayed stating that field is required. The biggest challenge on the Web App was how to integrate the 3D view of locations. As mentioned before in the previous chapter this was achieved using the WebGL API and the game engine Unity. To integrate the WebGL in the web pages an inline frame was used, specifically on the Assignment creation page and the Assignments list page. These two pages are the only ones with the 3D view since here is where it is most needed and for performance purposes. The loading of the WebGL is impacted by the performance of the server where it is deployed and since this project is running on the lowest Azure plan it can take some seconds to load. Since the WebGL project makes web requests to the Services module it requires authentication and also the identifier of the location we want to display. In order to make it more user-friendly, the token from local storage created on the website login and the location identifier are automatically passed to the WebGL build using the local storage of the browser.

4.2.4 WebGL

As mentioned before, this module was built to have the 3D view on internet browsers represented by the Front-End module. Since both the Mobile App module and this one are built with the same game engine, Unity 2021.7f1, it was possible to reuse material from one project to the other saving some developing time. In reality, this is a derivative of the Mobile App since the Mobile App was the first to be built between the two. The WebGL module consists of 3D scenes of Venues, which are divisions of a Location, and since the main purpose of the website is to consult data the user interface was simplified compared to the Mobile App. To build the scenes, the 3D models created in the component modelling phase are imported into the Unity project. For an easier interaction with the scene user controls made available by the Unity community were used to move the player using the keyboard and mouse [33].

To access the Services, a WebRequest class was made for the generic Rest calls to avoid code repetition. Every service of the application that needs to make a call will pass through the WebRequest class. This class also has the task of Deserializing the JSON objects returned by the Services module. This way we avoid code repetition and have the conversion methods all in one place. Unity has objects called GameObjects that can have scripts attached to them to perform a certain behaviour. For the API calls an object that is global to all scenes, meaning is not destroyed when changing between them, was created with the WebRequest and Service scripts attached to it. This allows preserving data between scenes, for example, the token and previously downloaded data to prevent the same request to the Services module.

Figure 4.5: The Unity Editor view of a scene.

In figure 4.5 we can observe the Unity Editor, on the left the GameObjects of the

current scene, the Venue with identifier 1, displayed in the middle and on the right the web request scripts attached to the controller GameObject. The scene naming is important for its loading during use by the user since the scene loader expects the identifier to be in the name. To give an example, for the scene in figure 4.5 to be displayed on either the mobile or the website the LoadVenueData script takes the identifier and searches for a scene file name with the format Scene_Identifier. For this reason, the venue must be added to the database first using the website and after that implemented in Unity.

For easier visualisation of the inspection location on a specific component the idea of coloured Dots, small spherical objects on the components, emerged. With this, the inspector and client can have a better perspective of the position where that data was taken from the component. The colours, for now, Green, Yellow and Red, indicate the severity of the values registered, from within specification limits, to approaching limits and outside specifications respectively. Currently, the intervals for the measurement values that each colour represents need to be established by the client outside the context of this system. And so the inspector manually applies this colour to each inspection and the Dot takes the colour of the latest inspection. Examples of the selection, user interface and the Dots on the components will be displayed in the results chapter.

Unity has an object type called prefab, which is a generic object that can be used in multiple scenes. This way if we need to change the object only the prefab needs to be updated and the change is applied to all scenes using it. This also allows faster development of new venues by simply dragging the needed prefab and respective script onto a new scene. To create, view or edit the object on the scene a simple user interface was made.

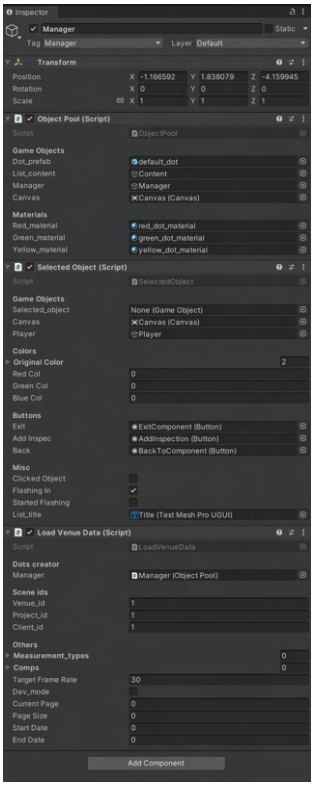

Figure 4.6: Scripts attached to the Manager GameObject.

The display of the Interface, selection of components and creation of points is managed by the scripts attached to the Manager GameObject, figure 4.6. The ObjectPool script is responsible for the creation of Dots on the selected component, the SelectedObject script handles the selection of the component when the user clicks on it and the display of the user interface with the list of Dots of that component. Finally, the LoadVenueData, when a scene of a Venue is loaded this script loads the data for each component, Dot and inspection using the web requests.

All features mentioned so far are valid for both this module and the mobile module with very small differences in the user interface and user controls. The user controls will differ because the mobile version needs to use a touchscreen approach for moving the camera. The WebGL API uses JavaScript language which does not support multithreading, this was a challenge because it prevented the application from performing parallel tasks. An example would be making a web request in the background while performing other tasks and at the same time waiting for a response on the web request. A situation like this would freeze the application, and to overcome this problem the logic needs to be executed over several frames. Unity has a function called Coroutine that allows you to pause a function and tell it to wait for a condition or action to occur before continuing. This way functionality can be split up into several steps that can be executed in order[34]. Using Coroutines fixed the problem and waiting for a Web Request no longer freezes the WebGL application.

4.3 Mobile Layer

This layer corresponds to the mobile application used mainly by inspectors on locations to view and register inspection data.

4.3.1 Mobile App

The core functionalities and implementation were mentioned on the WebGL module since as mentioned they are both built on the Unity game engine. What varies in this module is the more robust user interface, being a separate application it needed a login page and some way to inform the inspector about the assignments.

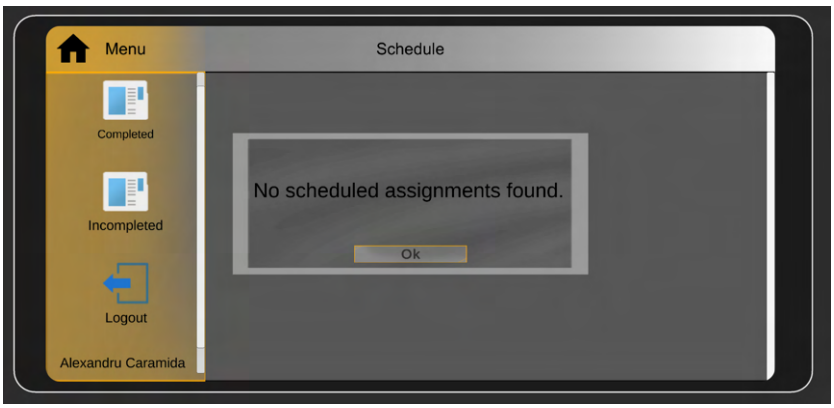

Figure 4.7: The Schedule page with the sidebar menu opened.

This was achieved by creating scenes to list the assignments, in three types, the assignments schedules for the current day, the ones that remain incomplete and the ones that were already complete. The user can navigate between these lists using a sidebar menu, figure 4.7.

Figure 4.8: The sidebar menu opened with assignment status.

One other addition to the user interface is the ability to see what components are marked for inspection in the current assignment and the assignment status in the sidebar

menu as seen in figure 4.8. In the inspection form, the ability to upload a picture was also added. This feature was achieved using an Asset from the Unity Store, this plugin [35] helps show a file browser to choose the desired picture from the device file system.

As for player movement, the Mobile App needed a different approach since a keyboard and mouse are not an option. For this, another community-made movement script intended for mobile screens was used [36]. The script works by dividing the screen into two equal parts, defining the left as the movement touch and the right as the rotation. This way the user can navigate through the scene as if it was using a game controller with joysticks. The movement scripts were sourced from the community to spend more time focusing on the core features of the 3D and other modules.

5

Results

As mentioned before, the interaction with the final users was weak and so the users experience can not reported here. Nonetheless we present two stories that highlight the future use of this system. The web browser used was Microsoft Edge on computers running Windows 10 for the back office, which currently is accessed via the link https://tesebackendwebsite45604.azurewebsites.net/. The Mobile App was used with a Samsung Galaxy Note9 device running Android 10.

5.1 User Story One

This story is composed of three parts, the venue in this story used objects from the community. In the first one, the manager creates an assignment for a technician on one location of the client **Navigator**, specifically the venue **B** on "Avenida Noruega 9, 5000-101 Vila Real Portugal". The technician proceeds to the assignment in part two. Finally, in part three the client views the assignment and the inspection data registered by the technician.

5.1.1 Part 1

The **Manager** creates an **Assignment** to inspect the venue "B" of a location associated with the client **Navigator**, using the web browser Microsoft Edge.

1. Access the Backoffice site and login with the admin account, username "a.caramida" password "password".
2. On the client "Navigator" card click on "Projects".
3. Click on "Locations" of the project card "Paper".
4. Next click on "Venues" of the location with the address "Avenida Noruega 9, 5000-101 Vila Real Portugal".
5. Once on the venues page click on "Assignments" of venue "B".

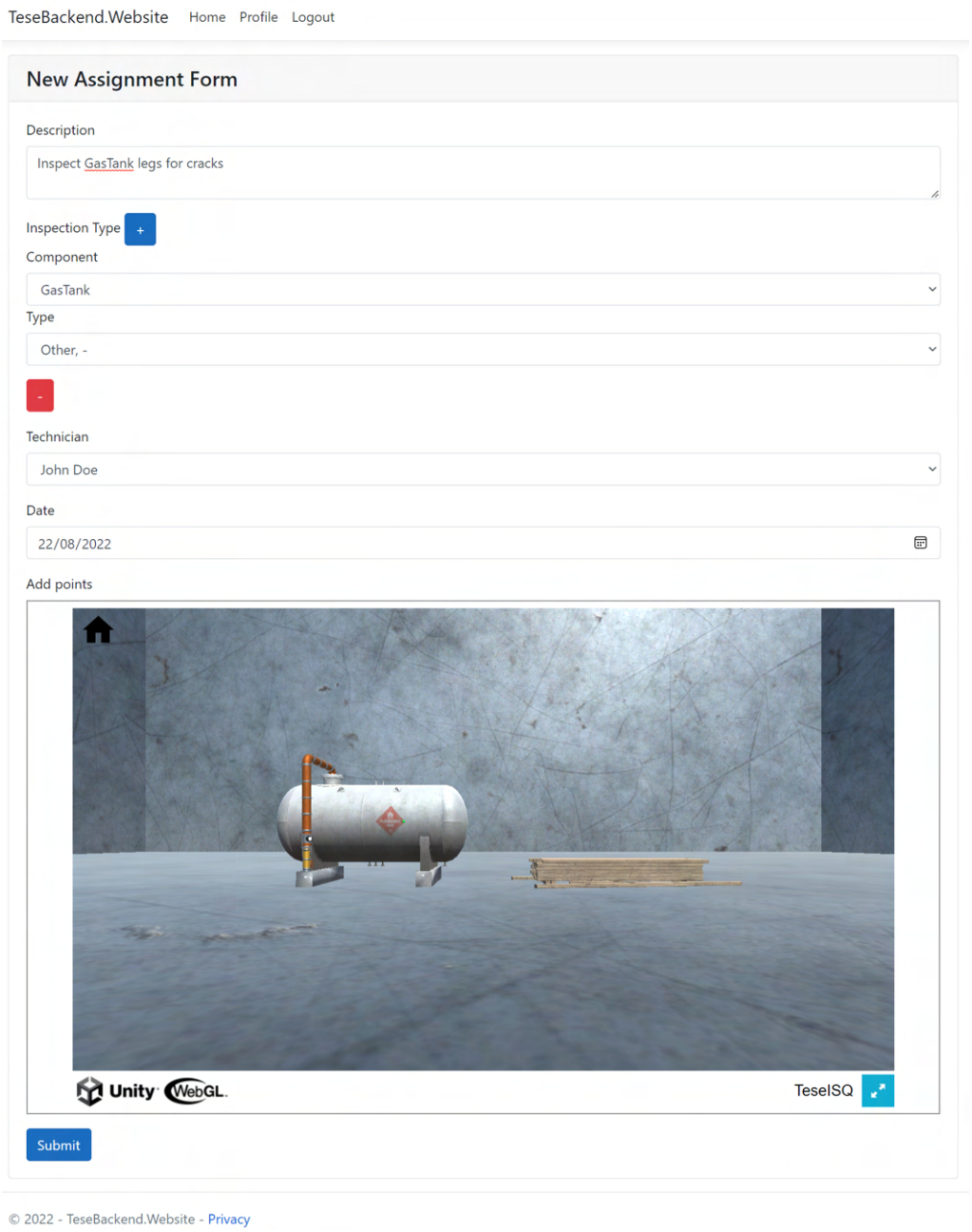

Figure 5.1: Page with the form to add a new Assignment.

6. To create a new assignment click on “Add Assignment”, which takes you to the form seen in figure 5.1.

7. Write the description “Inspect GasTank legs for cracks”.

8. Click on “+” and choose the component “GasTank” and the type “Other”.

9. Attribute the assignment to technician “John Doe” and set the assignment date for today.

10. Next interact with the 3D view of the venue, using the WASD keys and the mouse, and select the "GasTank" with a double click on the mouse left button.

11. Once selected use the mouse right click to create a "Dot" on each leg of the "GasTank" and the descriptions "front leg" and "back leg".

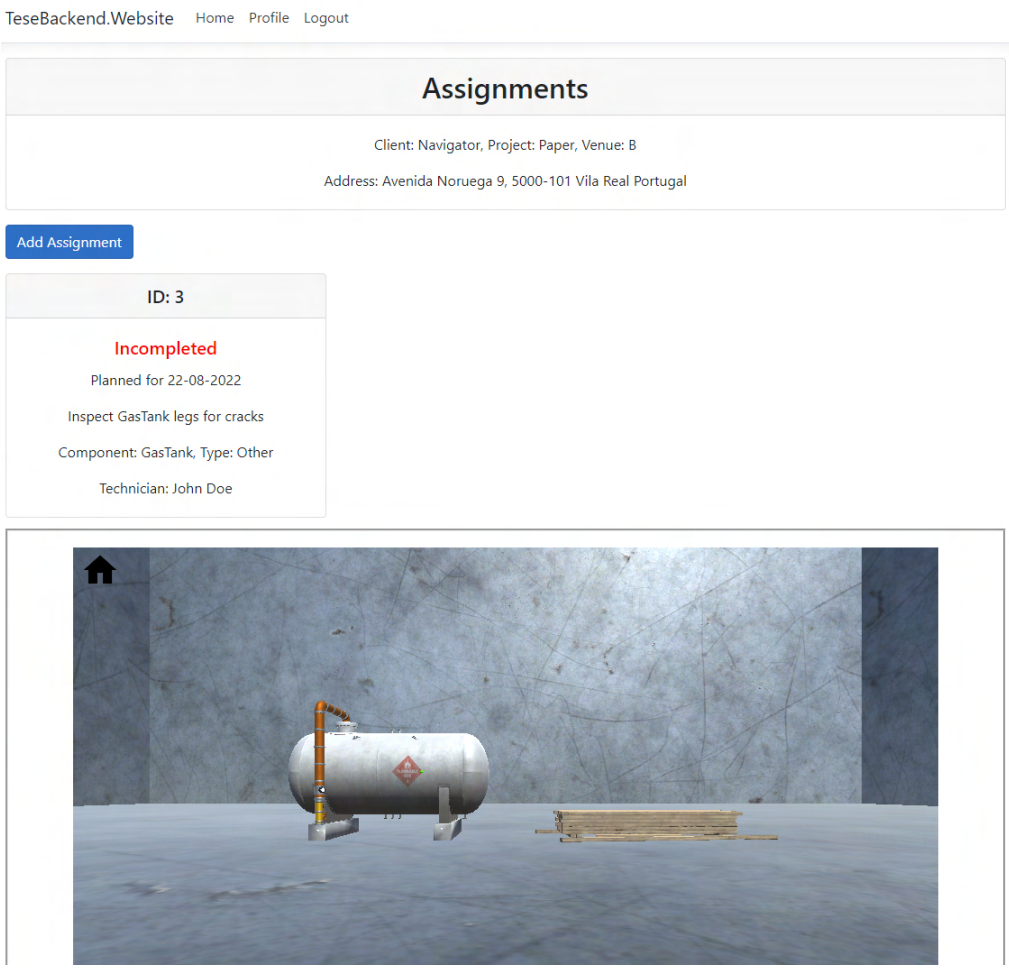

Figure 5.2: Assignments page after creating the new assignment.

12. Next click "Submit" to save and be redirected to the assignments page as seen in figure 5.2.

5.1.2 Part 2

The **Technician** chosen in part 1 proceeds to the location to carry on the assignment, this story assumes the location has Internet access and the device is an Android tablet in landscape mode.

1. Access the app using the device and the technician account, username "j.doe" password "password".

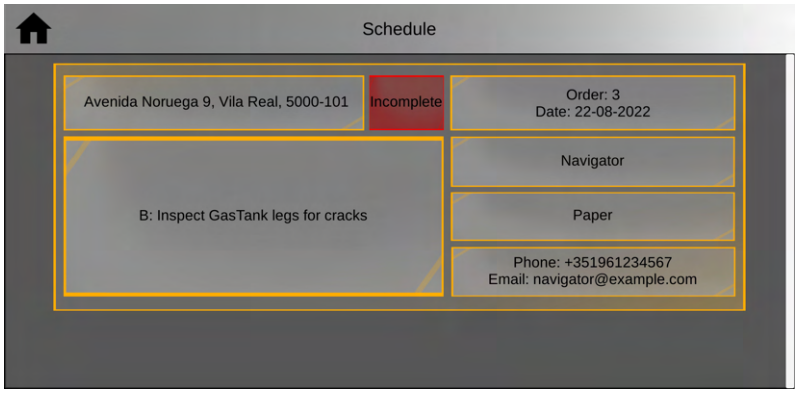

Figure 5.3: Schedule page on the mobile application.

2. On the schedule page, figure 5.3, look up the address for the "Navigator" assignment and go to the address listed on the assignment.
3. Once on-site access the application and click on the description of the assignment.
4. Now with the 3D view of the venue click on the menu button on the upper left corner of the screen to open the sidebar.
5. On the sidebar menu click on "Components to inspect".

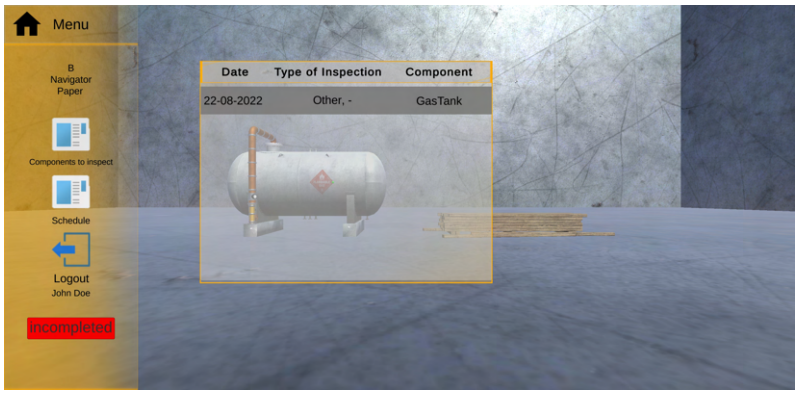

Figure 5.4: The list of components to inspect in this assignment.

6. The table with the components and type of inspection for this assignment are shown, for example in figure 5.4, now click on the component "GasTank".

Figure 5.5: The list of Dots of the component GasTank at that time.

7. On the right side of the screen the list of "Dots", points of interest of the component, appears as seen in figure 5.5.

8. Click on the "Severity" of a Dot with "None", in this case, the ones created previously by the Manager.

9. The list changes to the inspections registered on that Dot, click on the "+" symbol on the upper left corner of the list.

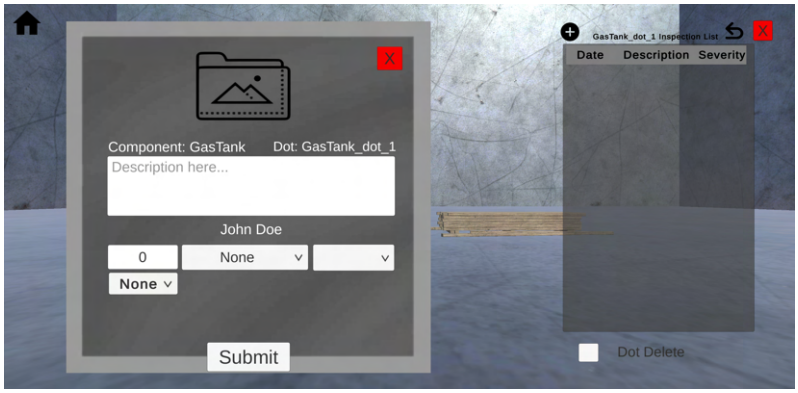

Figure 5.6: The inspection form.

10. A form to fill the inspection data is shown, as seen in figure 5.6.

11. Switch to the camera application of the device and take a picture of the component on the approximate location of the Dot. Return to the Mobile App and continue the inspection.

12. Add the description of said inspection, set the measurement under the technician's name to "Other", and the severity under the value to "Green".

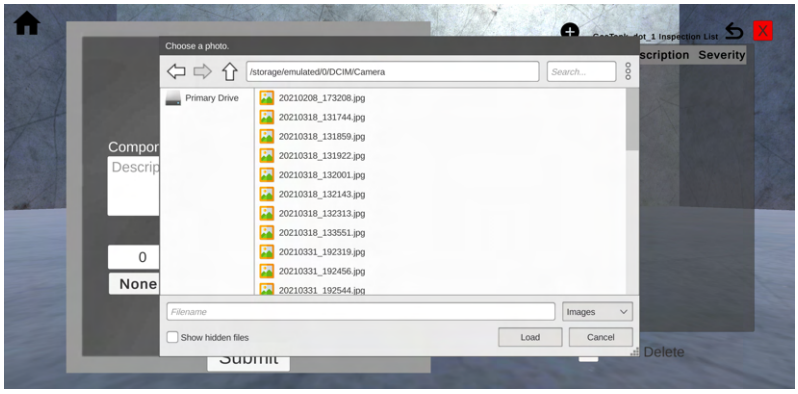

Figure 5.7: File browser to choose a photo.

13. Click on the folder picture and choose the picture taken moments ago using the file browser, for example in figure 5.7.
14. Lastly click on submit to save the inspection data.
15. Repeat this step for the second Dot of the selected component.
16. Once all Dots are inspected access the sidebar menu again and click the red button at the bottom to change the status of the assignment to “Completed”.

5.1.3 Part 3

A user of the client **Navigator** accesses the website to view the assignment status.

1. Access the Backoffice site with the client account, username “j.smith” password “password”.
2. Click on “Locations” of the project card “Paper”.
3. Next click on “Venues” of the location with the address “Avenida Noruega 9, 5000-101 Vila Real Portugal”.
4. Once in the venues page click on “Assignments” of the venue “B”.
5. Next interact with the 3D view of the venue, using the WASD keys and the mouse, and select the “GasTank” with a double click on the mouse left button.

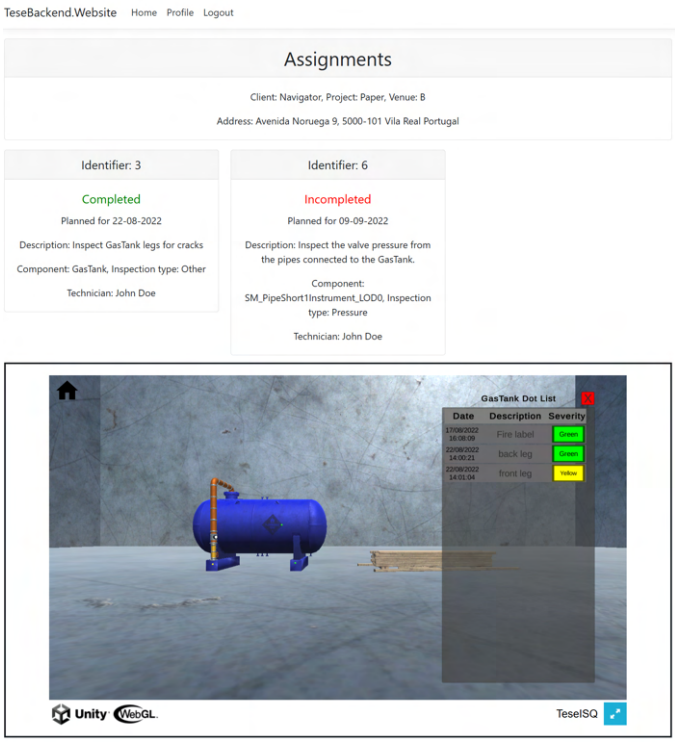

Figure 5.8: GasTank selected and its list of Dots at that time.

6. On the right side of the screen the list of “Dots” appears as seen in figure 5.8.
7. Click on the “Severity” of the Dot with the description "back leg"to access its inspections.
8. The list changes to the inspections registered on that Dot, click on the “Severity” of an inspection to view its details.

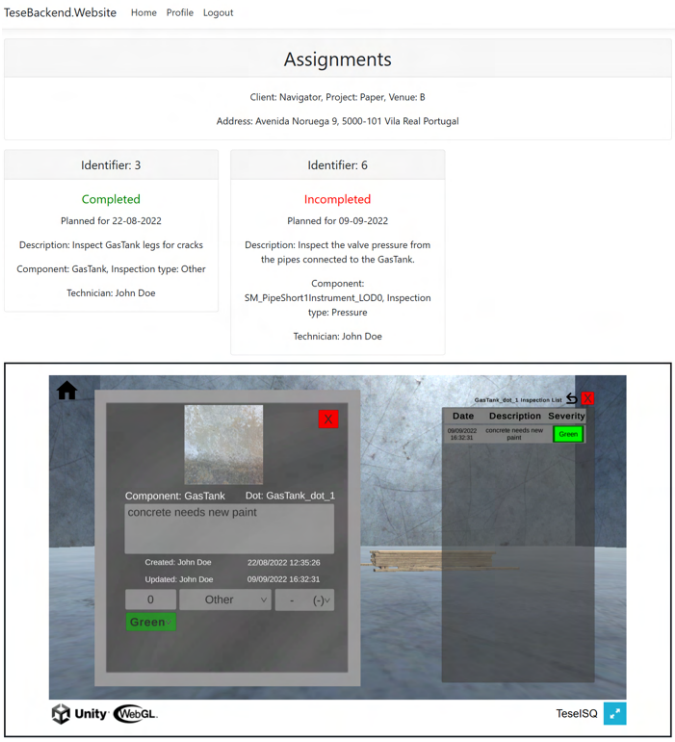

Figure 5.9: The view of an inspection of the selected Dot.

9. A page with the inspection data is shown as seen in figure 5.9.

5.2 User Story Two

This story is composed of three parts, the venue of this story has models made with Scaniverse. There will be some task variations to show there is more than one way of doing them. In the first one, the manager creates an assignment for a technician on one location of the client **Navigator**, specifically the venue **A** on "Avenida Noruega 9, 5000-101 Vila Real Portugal". The technician proceeds to do the assignment in part two to proceed to the inspection. Finally, in part three the client views the assignment and the inspection data registered by the technician.

5.2.1 Part 1

The **Manager** creates an **Assignment** to inspect the venue "A" of a location associated with the client **Navigator**, using the web browser Microsoft Edge.

1. Access the Backoffice site and login with the admin account, username "a.caramida" password "password".
2. On the client "Navigator" card click on "Projects".
3. Click on "Locations" of the project card "Paper".

4. Next click on “Venues” of the location with the address “Avenida Noruega 9, 5000-101 Vila Real Portugal”.
5. Once on the venues page click on “Assignments” of the venue “A”.
6. To create a new assignment click on “Add Assignment”.
7. Write the description “Inspect Car tires pressure”.
8. Click on “+” and choose the component “Car” and the type “Pressure”.
9. Attribute the assignment to technician “John Doe” and set the assignment date for today.
10. Click “Submit” to conclude the assignment creation.

5.2.2 Part 2

The **Technician** chosen in part 1 proceeds to the location to carry on the assignment, this story assumes the location has Internet access and the device is an Android tablet in landscape mode.

1. Access the app using the device and the technician account, username “j.doe” password “password”.
2. On the schedule page look up the address for the “Navigator” assignment and go to the address listed on the assignment.
3. Once on-site access the application and click on the description of the assignment.
4. Now with the 3D view of the venue click on the menu button on the upper left corner of the screen to open the sidebar.
5. On the sidebar menu click on “Components to inspect”.
6. The table with the components and type of inspection for this assignment are shown, click on the component “Car”.
7. On the right side of the screen the list of “Dots”, points of interest of the component, appears.
8. Click on the “Severity” of a Dot with tire description, if not all tires have Dots create them using the right mouse click on the desired spot and add a description to identify them.
9. The list changes to the inspections registered on that Dot, click on the “+” symbol on the upper left corner of the list.
10. A form to fill the inspection data is shown, proceed with the inspection of the Dot.

11. Switch to the camera application of the device and take a picture of the component on the approximate location of the Dot. Return to the Inspection application to register the data.

12. Add the description of said inspection, set the measurement under the technician's name to "Pressure", set the value, and the severity under the value to "Green".

Figure 5.10: The inspection form completed.

13. Click on the folder picture and choose the picture taken moments ago, an example of the photo chosen in figure 5.10.

14. Lastly click on submit to save the inspection data.

15. Repeat these steps for all tires of the "Car".

Figure 5.11: The assignment set as completed and the Car Dot list.

16. Once all is done access the sidebar menu again and click the red button at the bottom to change the status of the assignment to "Completed" as seen in figure 5.11.

5.2.3 Part 3

A user of the client **Navigator** accesses the website to view the assignment status.

1. Access the Backoffice site with the client account, username "j.smith" password "password".
2. Click on "Locations" of the project card "Paper".
3. Next click on "Venues" of the location with the address "Avenida Noruega 9, 5000-101 Vila Real Portugal".
4. Once on the venue page click on "Assignments" of the venue "A".
5. Next interact with the 3D view of the venue, using the WASD keys and the mouse, and select the "Car" with a double click on the mouse left button.

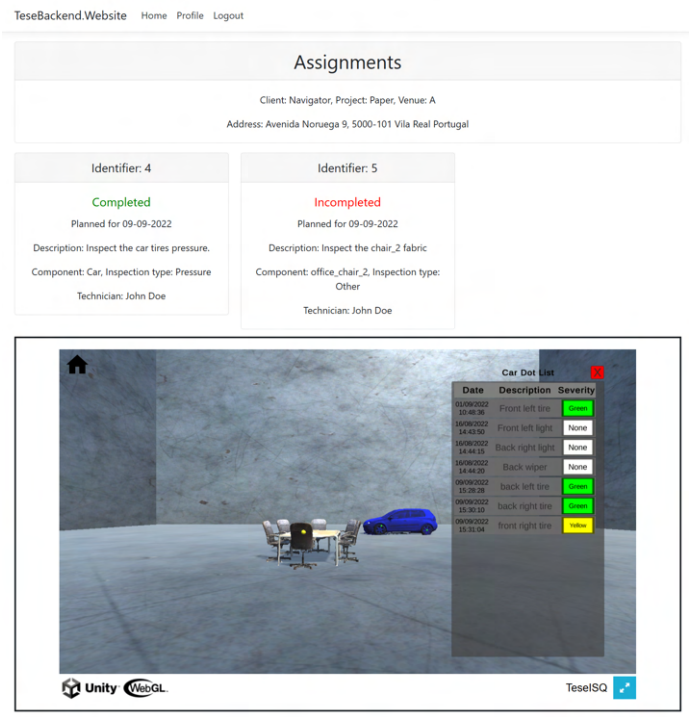

Figure 5.12: Car selected and its list of Dots at that time.

6. On the right side of the screen the list of "Dots" appears, points of interest of the component as seen in figure 5.12.
7. Click on the "Severity" of a Dot to access its inspections.
8. The list changes to the inspections registered on that Dot, click on the "Severity" of an inspection to view its details.
9. A page with the inspection data is shown.

6

Conclusions

6.1 Conclusions

After testing the system with the user stories, we could say that having a 3D visualisation of the industrial environment can help the inspection process. Unfortunately, the difference and impact the system could potentially have in an actual inspection situation could not be witnessed since it was not possible to visit or witness the process of an inspection. Besides having a visual of the environment, the system also organises the inspection data in a structured database for easier and more organised management. It can also be said that the use of the system can facilitate the organising of data when compared with unstructured or paper. However, there is still a need for an onsite test to verify the reliability of the modelling process for industrial environments, as these sites usually have very large components.

In the end, it was not possible to achieve all objectives, namely the positioning using georeferencing, but registering inspection data with an 3D visualisation of the environment was achieved.

6.2 Future Work

Although the core basics are present some features could not be developed during this dissertation time. This section mentions some of these objectives that need to be implemented and other possible improvements to the system.

The addition of a filtering system for the data would help users to have easier and more efficient access to data. Filtering the inspections for example by date or by severity could help better understand the status of a component over time. Besides filtering data, completing the pagination of lists on the user interface is also necessary.

The client currently has a large amount of data and so the ability to upload data in bulk (e.g. spreadsheets) would help transition from the old system to the new one. Add a georeferenced coordinate system to the 3D to position the components on the scene according to their position in the environment. This could help make the creation

and editing of the 3D scene easier by automatically placing the components instead of a developer having to place them manually.

Add a persistent caching system to the Mobile App for when the internet connection is not reliable or available. The inspector will be able to perform his task with no problem and once the connection is reestablished the system uploads the new data.

Finally an automated warning system for comparing inspection data with past references and notifying the user about concerning values. As well as automatically attributing a severity colour when registering the measurement values.

Bibliography

[1] João M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[2] sansmachining. *3D Compare To 2D CAD Drawing, What Is The Difference And Advantages?* [Online; accessed 31-January-2022]. URL: https://www.sansmachining.com/3d-compare-to-2d-cad-drawing-what-is-the-difference-and-advantages/ (visited on 01/31/2022) (cit. on pp. 4, 5).

[3] Carlos Martínez. *Make 3D models from photos*. [Online; accessed 09-February-2022]. URL: https://meshroom.en.uptodown.com/windows (visited on 02/09/2022) (cit. on p. 5).

[4] Scaniverse. *Capture life in 3D*. [Online; accessed 27-August-2022]. URL: https://scaniverse.com/ (visited on 08/27/2022) (cit. on p. 5).

[5] Blender Foundation. *About Blender*. [Online; accessed 31-January-2022]. URL: https://www.blender.org/about/ (visited on 01/31/2022) (cit. on p. 6).

[6] Blender Foundation. *The Freedom to Create*. [Online; accessed 13-September-2022]. URL: https://www.blender.org/about/ (visited on 09/13/2022) (cit. on p. 6).

[7] Blender Foundation. *Developer Documentation*. [Online; accessed 09-February-2022]. June 2019. URL: https://wiki.blender.org/wiki/Main_Page (visited on 02/09/2022) (cit. on p. 6).

[8] Kai Inge Midtgård Rokstad. *What is SolidWorks?* [Online; accessed 13-September-2022]. URL: https://www.technia.com/blog/what-is-solidworks/ (visited on 09/13/2022) (cit. on p. 6).

[9] Artec 3D. *Optimal scan-to-CAD solutions*. [Online; accessed 31-January-2022]. URL: https://www.artec3d.com/3d-software/solidworks-cad (visited on 01/31/2022) (cit. on p. 6).

[10] Artec 3D. *Artec Scanning SDK 2.0*. [Online; accessed 31-January-2022]. URL: https://www.artec3d.com/3d-software/sdk (visited on 01/31/2022) (cit. on p. 7).

[11] Artec 3D. *Artec Ray 3D Scanner*. [Online; accessed 09-February-2022]. URL: https://www.artec3d.com/portable-3d-scanners/laser-ray (visited on 02/09/2022) (cit. on p. 7).

[12] Educba. *What is Unreal Engine?* [Online; accessed 13-September-2022]. URL: https://www.educba.com/what-is-unreal-engine/ (visited on 09/13/2022) (cit. on p. 8).

[13] Evelyn Trainor-Fogleman. *Unity vs Unreal Engine: Game engine comparison guide for 2021*. [Online; accessed 31-January-2022]. URL: https://www.evercast.us/blog/unity-vs-unreal-engine (visited on 01/31/2022) (cit. on p. 8).

[14] Program-Ace. *Unity vs. Unreal: What to Choose for Your Project?* [Online; accessed 31-January-2022]. Apr. 2021. URL: https://program-ace.com/blog/unity-vs-unreal/ (visited on 01/31/2022) (cit. on pp. 8, 9).

[15] freeCodeCamp. *Unity Game Engine Guide: How to Get Started with the Most Popular Game Engine Out There*. [Online; accessed 13-September-2022]. Feb. 2020. URL: https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/ (visited on 09/13/2022) (cit. on pp. 8, 9).

[16] Greg Corke. *Unity for manufacturing*. [Online; accessed 10-February-2022]. Oct. 2019. URL: https://develop3d.com/features/unity-visualisation-vr-manufacturing-industrial-design-game-on-simulation/ (visited on 02/10/2022) (cit. on p. 9).

[17] Jead - Unity Technologies. *What subscription tiers are available?* [Online; accessed 31-January-2022]. URL: https://support.unity.com/hc/en-us/articles/208610336-What-subscription-tiers-are-available- (visited on 01/31/2022) (cit. on p. 9).

[18] Xiaobai A. Yao. *Georeferencing and Geocoding*. Ed. by Audrey Kobayashi. Second Edition. Oxford: Elsevier, 2020, pp. 111–117. ISBN: 978-0-08-102296-2. DOI: https://doi.org/10.1016/B978-0-08-102295-5.10548-7. URL: https://www.sciencedirect.com/science/article/pii/B9780081022955105487 (cit. on pp. 9, 10).

[19] Esri. *Spatial references in geodatabases*. [Online; accessed 09-February-2022]. URL: https://desktop.arcgis.com/en/arcmap/latest/manage-data/geodatabases/the-properties-of-a-spatial-reference.htm (visited on 02/09/2022) (cit. on p. 10).

[20] Esri. *Vertical coordinate systems*. [Online; accessed 09-February-2022]. URL: https://desktop.arcgis.com/en/arcmap/latest/map/projections/fundamentals-of-vertical-coordinate-systems.htm (visited on 02/09/2022) (cit. on p. 10).

[21] MaintainX. *Everyone loves downtime, just not at your plant.* [Online; accessed 11-February-2022]. URL: https://www.getmaintainx.com/industries/manufacturing/ (visited on 02/11/2022) (cit. on p. 11).

[22] Capptions. *Inspection Software to enhance your Safety Processes.* [Online; accessed 11-February-2022]. URL: https://www.capptions.com/inspection-software?utm_source=SoftwareAdvice (visited on 02/11/2022) (cit. on p. 12).

[23] StatCounter. *Mobile Android Version Market Share Worldwide.* [Online; accessed 05-September-2022]. URL: https://gs.statcounter.com/android-version-market-share/mobile/worldwide/ (visited on 09/05/2022) (cit. on p. 18).

[24] Scaniverse. *Scanning Tips.* [Online; accessed 06-September-2022]. URL: https://scaniverse.com/support (visited on 09/06/2022) (cit. on p. 21).

[25] Microsoft. *Azure App Service.* [Online; accessed 06-September-2022]. URL: https://azure.microsoft.com/en-gb/services/app-service/#overview (visited on 09/06/2022) (cit. on p. 22).

[26] SQLite Consortium. *About SQLite.* [Online; accessed 06-September-2022]. URL: https://www.sqlite.org/about.html (visited on 09/06/2022) (cit. on p. 22).

[27] DB Browser for SQLite. *The Official home of the DB Browser for SQLite.* [Online; accessed 08-September-2022]. URL: https://sqlitebrowser.org/ (visited on 09/08/2022) (cit. on p. 23).

[28] Microsoft. *ASP.NET Core 101.* [Online; accessed 06-September-2022]. URL: https://docs.microsoft.com/en-us/shows/ASPNET-Core-101/?WT.mc_id=Educationaspnet-c9-niner (visited on 09/06/2022) (cit. on pp. 24, 25).

[29] Microsoft. *ASP.NET Core.* [Online; accessed 06-September-2022]. URL: https://github.com/dotnet/aspnetcore (visited on 09/06/2022) (cit. on p. 24).

[30] Microsoft. *Entity Framework Core.* [Online; accessed 08-September-2022]. URL: https://docs.microsoft.com/en-us/ef/core/ (visited on 09/08/2022) (cit. on p. 25).

[31] Auth0. *Introduction to JSON Web Tokens.* [Online; accessed 09-September-2022]. URL: https://jwt.io/introduction/ (visited on 09/09/2022) (cit. on p. 25).

[32] Microsoft. *ASP.NET Core Blazor.* [Online; accessed 09-September-2022]. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0 (visited on 09/09/2022) (cit. on p. 25).

[33] ashleydavis. *Unity FreeCam.* [Online; accessed 09-September-2022]. URL: https://gist.github.com/ashleydavis/f025c03a9221bc840a2b#file-unity-freecam (visited on 09/09/2022) (cit. on p. 27).

[34] Unity Technologies. *Coroutines.* [Online; accessed 10-September-2022]. URL: https://docs.unity3d.com/Manual/Coroutines.html (visited on 09/10/2022) (cit. on p. 29).

[35] yasirkula. *Runtime File Browser*. [Online; accessed 09-September-2022]. URL: https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006 (visited on 09/09/2022) (cit. on p. 31).

[36] itsPeetah. *Basic mobile FPS and TPS controller*. [Online; accessed 09-September-2022]. URL: https://github.com/itsPeetah/Mobile-FPS-and-TPS-controller-for-Unity (visited on 09/09/2022) (cit. on p. 31).

I

Database tables

The annexed files provide an individual view of the Database tables.

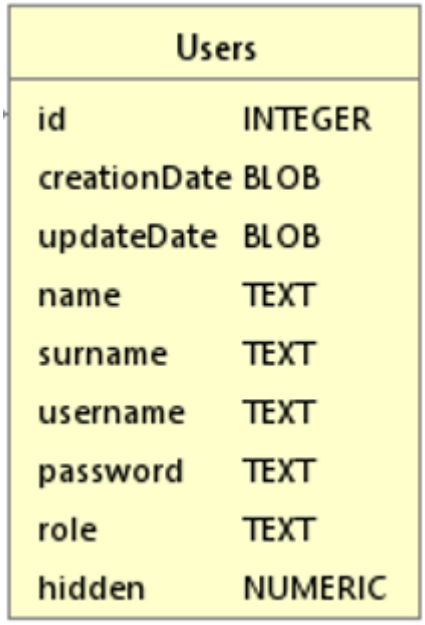

Figure I.1: The Users table.

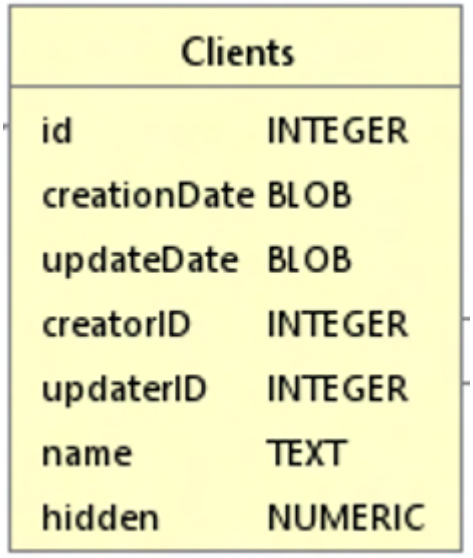

Figure I.2: The Clients table.

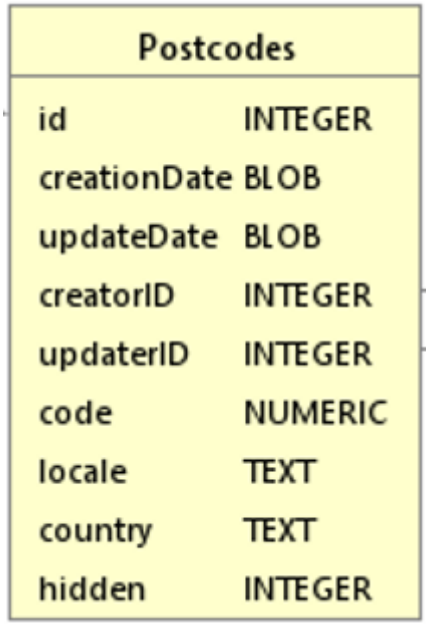

Figure I.3: The Postcodes table.

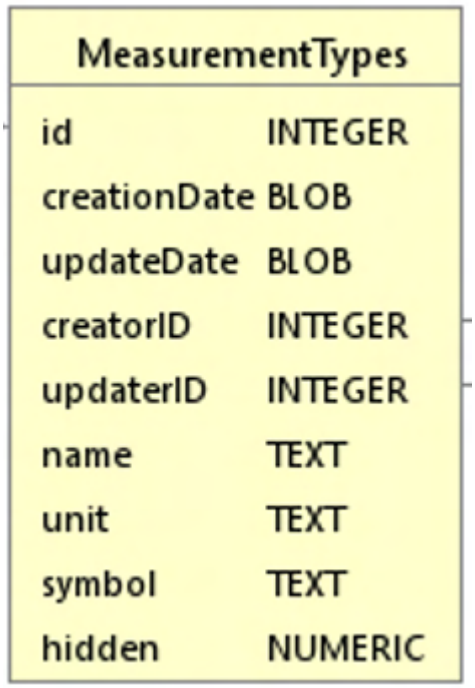

Figure I.4: The MeasurementTypes table.

Figure I.5: The ClientUsers table.

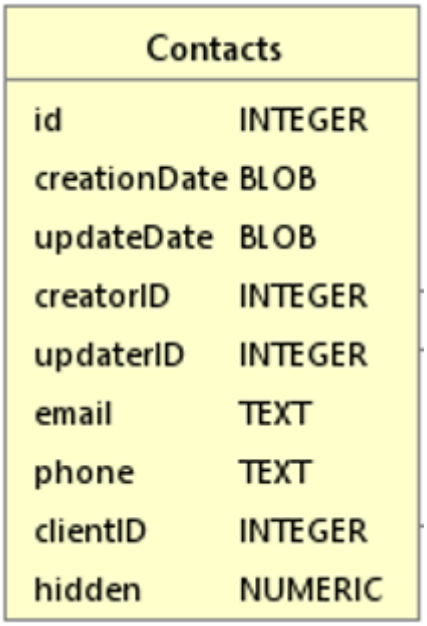

Figure I.6: The Contacts table.

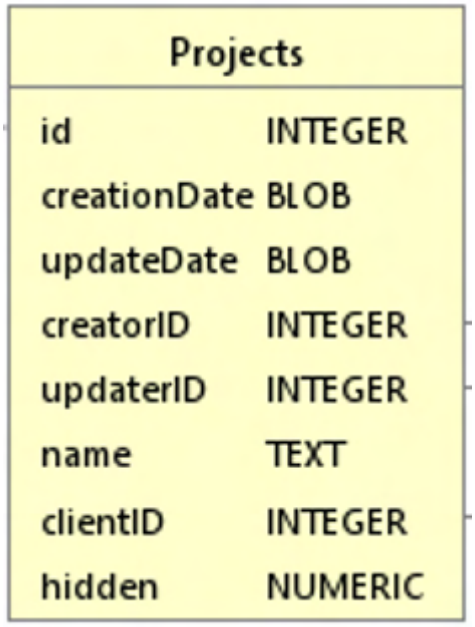

Figure I.7: The Projects table.

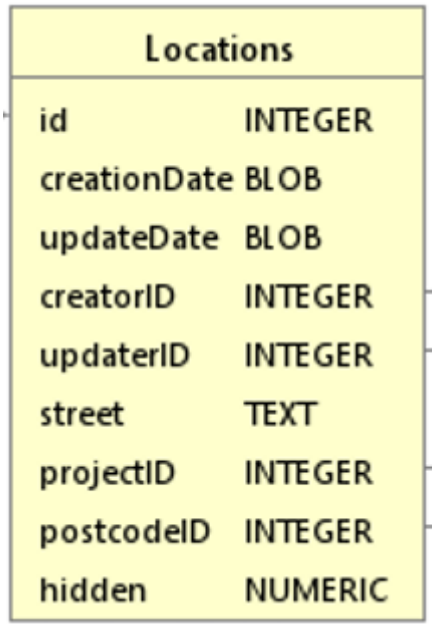

Figure I.8: The Locations table.

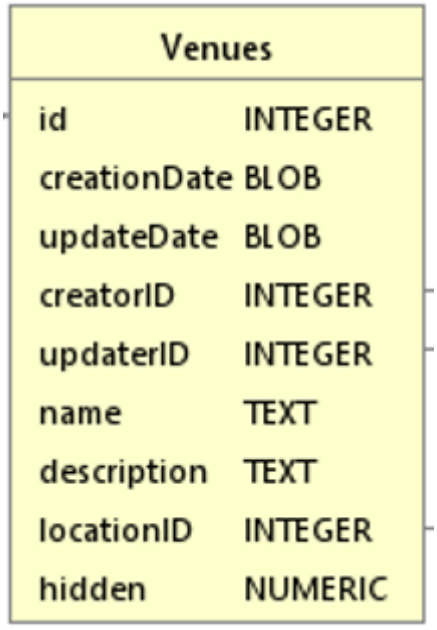

Figure I.9: The Venues table.

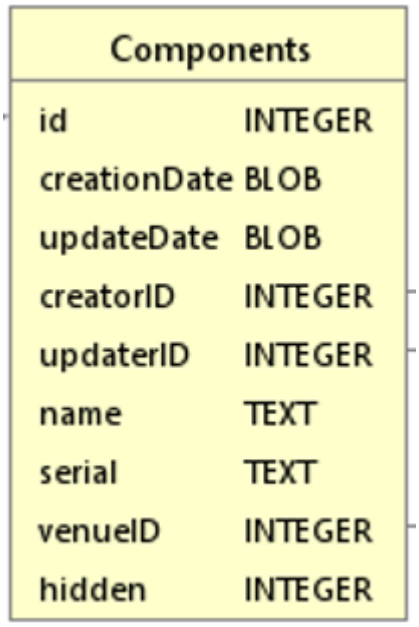

Figure I.10: The Components table.

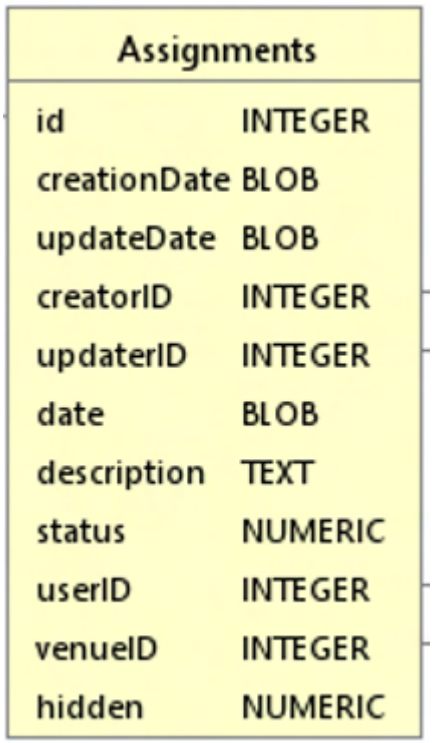

Figure I.11: The Assignments table.

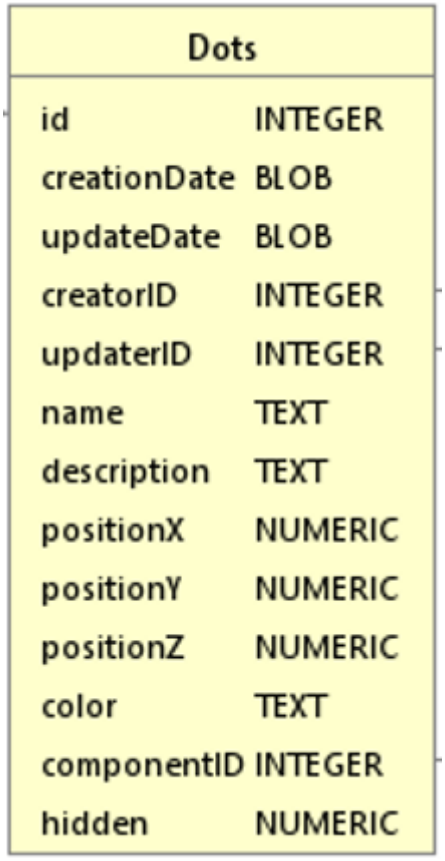

Figure I.12: The Dots table.

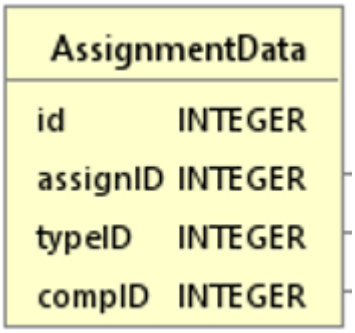

Figure I.13: The AssignmentData table.

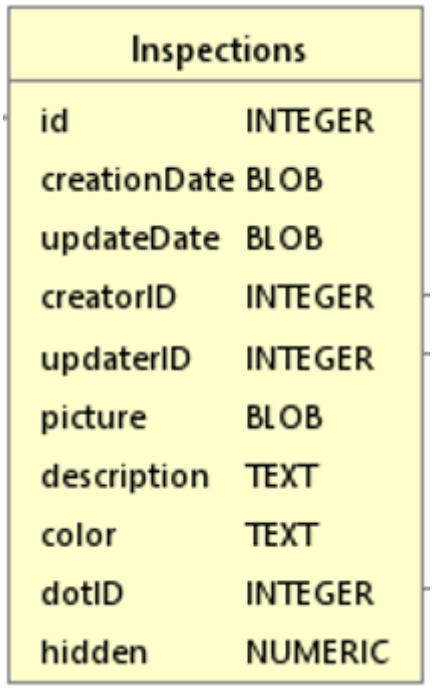

Figure I.14: The Inspections table.

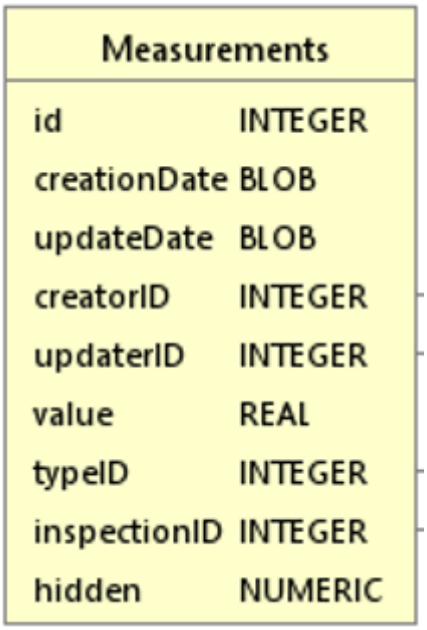

Figure I.15: The Measurements table.