Welcome to the Citiverse

Deliverable D 3.3

3D Environment Design Guidelines

Document Identification	
Project name:	Expanding the Experience of Citizens through Extended Reality (x-CITE)
Project number:	101167756
Duration:	December 2024 – May 2027
Delivery date:	3/11/2025
Responsible beneficiary:	Howest University of Applied Sciences
Contributing beneficiaries:	All consortium partners
Dissemination level:	Public
Version:	1.0
Status:	Final

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Document Information

Document Identification

Related WP	WP3	Related Deliverable(s)	D3.3
Document reference	D3.3_V1.0	Total number of pages	35

Document History

Version	Date	Contributor	Description
0.1	24/04/2025	Howest	Initial text descriptions for all chapters
0.2	21/08/2025	Howest	Added info about API's & Design frameworks
0.3	29/09/2025	Cegeka	Review round 1
0.4	03/10/2025	Howest	Implementation review round 1
0.5	03/10/2025	Howest	Added bibliography and performance matrix
0.6	17/10/2025	Future Insight	Review round 2
0.7	22/10/2025	Howest	Implementation review round 2
1.0	02/11/2025	Howest	Edits following consortium comments

Table of Contents

Document Information **2**
Document Identification 2
Document History 2

List of Tables **4**

List of Figures **4**

List of Acronyms **5**

Executive Summary **6**

WP and Task Targets **7**

Introduction *9*

Artistic requirements for immersive 3D environments *10*

3D formats for the project *12*
glTF (.gltf / .glb) 12
3D Tiles 13
FBX (.fbx) 14
OBJ (.obj) 14
When to use which 3D format 15

Requirements for 3D Environments *16*
Visual and Artistic Requirements 16
Technical Requirements 17
Efficiency and optimization for 3D assets and models 21

Design Framework *24*
Step 1) Agree on goals 24
Step 2) Define 3 core personas 24
Step 3) Set design principles 25
Step 4) Choose the right interaction model 26
Step 5) Organize spatial UI layout 27
Step 6) Choose your visual language 27
Step 7) Create your component library 29
Step 8) Think about accessibility and comfort 29
Step 9) User Acceptance Testing 29

Propose a secure and scalable dataservice API *31*
Components of a secure and scalable dataservice API 31
What types of data the does the API need to handle? 33

Bibliography **35**

List of Tables

Table 1: When to use which 3D format 15
Table 2: Target FPS, reference device specs, and performance tips by platform..... 17
Table 3: API data types 34

List of Figures

Figure 1: Screenshot of 3D Rotterdam 10
Figure 2: Screenshot of The Netherlands in 3D 11
Figure 3: First demo of the 3D traffic viewer tool for the Flemish use case 24
Figure 4: Persona example from Nielsen Norman Group 25
Figure 5: 3D Rotterdam works with a similar interaction model 26
Figure 6: Pokémon GO uses the camera feed a main backdrop..... 26
Figure 7: Quick Look AR lets you access AR experiences directly from the browser..... 26
Figure 8: Google Earth VR is a free VR experience. Google..... 27
Figure 9: Different visual iterations for the Rotterdam use case 28
Figure 10: Several Meta Horizon OS UI Set components..... 29
Figure 11: Meta gives useful tips for more inclusive VR apps 29
Figure 12: Proposed mockups for “Rotterdam use case - iteration 0” 30

List of Acronyms

Abbreviation /acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
CP	Consortium Partners
DLSS	Deep Learning Super Sampling: AI-powered upscaling technology from NVIDIA
FPS	Frames per second (framerate/refresh rate)
FSR	FidelityFX™ Super Resolution: AI-powered upscaling technology from AMD
Hz	Hertz (refresh rate)
LiDAR	Light Detection and Ranging: remote sensing method that uses pulsed lasers to measure distances accurately
LDT	Local Digital Twins
VR	Virtual Reality
XR	Extended Reality

Executive Summary

The x-CITE Deliverable D3.3, “3D Environment Design Guidelines,” sets out practical, project-wide rules for designing immersive 3D environments and the enabling data/API architecture for XR tools within CitiVerse. Anchored in Work Package 3 (M2–M15), the document’s aim is to make XR services interoperable with diverse Local Digital Twins (LDTs) by aligning design choices with open standards and the OASC MIMs Plus toolbox, and by identifying any gaps that impede smooth XR-LDT integration.

The guidelines open with artistic and usability requirements that prioritize recognizability, clarity, and a semi-realistic tone suited to constrained devices. They illustrate realism–performance trade-offs using city-scale examples and stress that visual quality must be balanced with comfort and device limits. Close collaboration with UX and API tasks is mandated so that environments can visually integrate static 3D models and real-time data from LDTs.

A core section helps teams “choose the right format”: glTF/GLB for portable, PBR-capable real-time assets; FBX for animation-rich pipelines; OBJ for simple static meshes; and OGC 3D Tiles for streaming large, geospatially aligned city scenes with built-in LOD. A matrix summarizes best-use cases, strengths, limits, and when to use each format.

Technical requirements target sustained comfort: roughly 90–120 FPS for standalone/PC VR and ~60 FPS for mobile AR, with concrete device guidance and optimization tactics. The document details polygon and texture reduction, mesh merging, baked lighting, async loading, occlusion/foveated rendering, and disciplined LOD workflows (including step-by-step LOD setup and on-device tuning). These practices are tied to leading vendor guidelines to help teams hit frame-time budgets across varied hardware.

Scalability, reusability, and cross-platform compatibility are treated as first-order design goals. The paper advocates modular scene construction (reusable street furniture, façades, etc.), standardized asset exports (glTF/FBX/OBJ), and multi-device testing to catch resolution, input, and visual inconsistencies. A compact, 10-step design framework steers goals, personas, interaction models (web, AR, VR), spatial UI layout, accessible visual language, component libraries, and pilot validation with live LDT data.

Finally, the deliverable proposes a secure, scalable dataservice API tailored to XR. It is interoperability-first (OGC API Features, 3D Tiles, SensorThings, glTF 2.0), XR-ready for streaming/LOD/tiling, and privacy-by-design (OAuth/OIDC, least privilege, TLS, encryption) with explicit attention to GDPR and NIS2 operational duties. Collectively, the package ensures CitiVerse XR tools are performant, interoperable, and governance ready.

WP and Task Targets

Work Package 3: CitiVerse federation: interoperability, design and architecture for XR

Duration: M2 - M15

Lead Beneficiary: FUTURE INSIGHT

The primary objective of WP3 is to establish common design principles and integration requirements for XR tools within the context of and interoperability with digital twins.

This work package aims to ensure that the XR tools developed throughout the project are user-friendly, reusable and interoperable with various Local Digital Twins or similar platforms, based on international open standards and MIMs compliances. Additionally, these design principles will be created in alignment with the LDT Toolbox design principles.

More specifically, the proposed solutions for relevant BBs related to Extended Reality, integration and data management from the LDT Toolbox development project will be taken into account for the design of the integrated architecture of the x-CITE project.

WP3 is designed to achieve these goals through user analysis, design principle, formulation and technical integration processes. In this WP, we define and perform state of the art solutions for “CitiVerse federation: interoperability, design and architecture for XR”. When the project is concluded successfully, these will be expanded as implementation work.

The specific WP objectives are the following:

(1) define the **essential functionalities and requirements for XR tools**, catering to diverse user needs and preferences in the project;

(2) create a **design framework for intuitive user interfaces (UI) and user experiences (UX)** based on comprehensive analyses to enhance user engagement;

(3) define secure & scalable data service requirements for storing and exchanging environmental designs, ensuring **compatibility between VR and AR platforms on the one hand and Local Digital Twins on the other;**

(4) establish **design and artistic requirements for XR tools**, setting standards for visually appealing and immersive 3D environments;

(5) ensure **interoperability and reusability of XR tools** by defining integration processes with data APIs, leveraging best practices, industry insights and well-known international open standards.

WP3 will collaborate closely with the other work packages to ensure consistent integration and alignment across the project.

Participants: TAMP, NOKIA, VLO, IMEC, CGK, HOW, FI:

Introduction

This task focuses on designing and developing 3D environments that will be used in XR (Extended Reality) tools within the CitiVerse project. These environments will provide the visual foundation for both Virtual Reality (VR) and Augmented Reality (AR) applications, making them essential for creating immersive and user-friendly experiences.

To support these 3D environments, we will define both visual and technical requirements. These include artistic guidelines to ensure high-quality and engaging designs, as well as technical standards to ensure performance, scalability, and long-term use across different XR platforms. A secure and scalable data service API will also be proposed, allowing these environments to connect with other parts of the system via web-based or standalone interfaces.

Well-designed 3D environments help citizens see their city clearly. Landmarks, streets, and public spaces are easy to recognize, while simple labels and overlays show live information like traffic, air quality, or events in context. This makes services more understandable and decisions more transparent, so people can explore, plan, and give feedback with confidence. Clear visuals, readable text, and accessible controls ensure everyone can take part, on a phone, a browser, or a VR headset.

Another key focus of this task is making sure XR tools can work smoothly with Local Digital Twins (LDTs). We will analyze the LDT Toolbox reference architecture and its use of standards like OASC MIMs Plus to identify how well current systems support XR needs. Where gaps exist, we will provide recommendations to ensure better alignment and interoperability.

The outcome of this task will be a clear set of guidelines for creating 3D environments that are visually appealing, technically sound, and compatible with the broader CitiVerse ecosystem.

Artistic requirements for immersive 3D environments

To support the development of immersive and visually consistent 3D environments in the CitiVerse project, this subtask focuses on defining the artistic requirements that will guide asset creation across all XR tools. These requirements will form the basis for a future style guide that ensures a coherent visual identity, optimizes for performance, and supports cross-platform interoperability.

A core requirement is the definition of a consistent **art direction** that applies across all environments and assets. This includes selecting appropriate color palettes, material types, lighting styles, and environmental themes. The visual tone should support both realism and usability, maintain clarity and readability while still feeling believable and immersive. For example, urban environments in CitiVerse might follow a semi-realistic aesthetic that prioritizes recognizability and user orientation over photorealism, particularly on performance-constrained devices. A choice needs to be made in how realistic we want the visual components to be. Different examples can be used to give an idea of what is possible. For example, in the case of 3D Rotterdam, the choice has been made to use basic 3D shapes without realistic textures. This decision was taken mainly to ensure high performance and smooth interaction, as adding detailed textures would significantly increase the required computing power and loading times. In addition, detailed information about all buildings is not always available, which makes it difficult to apply realistic textures consistently across the whole city model.



Figure 1: Screenshot of 3D Rotterdam

In the case of The Netherlands in 3D a different approach is used, where the 3D buildings are textured as seen in the image below. This choice was made because textured models make buildings more recognizable and improve contextual accuracy, which is especially important in applications where orientation and realism matter. However, this also comes with a trade-off: using textures requires significantly more data and processing power to render, which can impact performance compared to simpler models. Those same principles from these Digital Twins apply to XR tools.

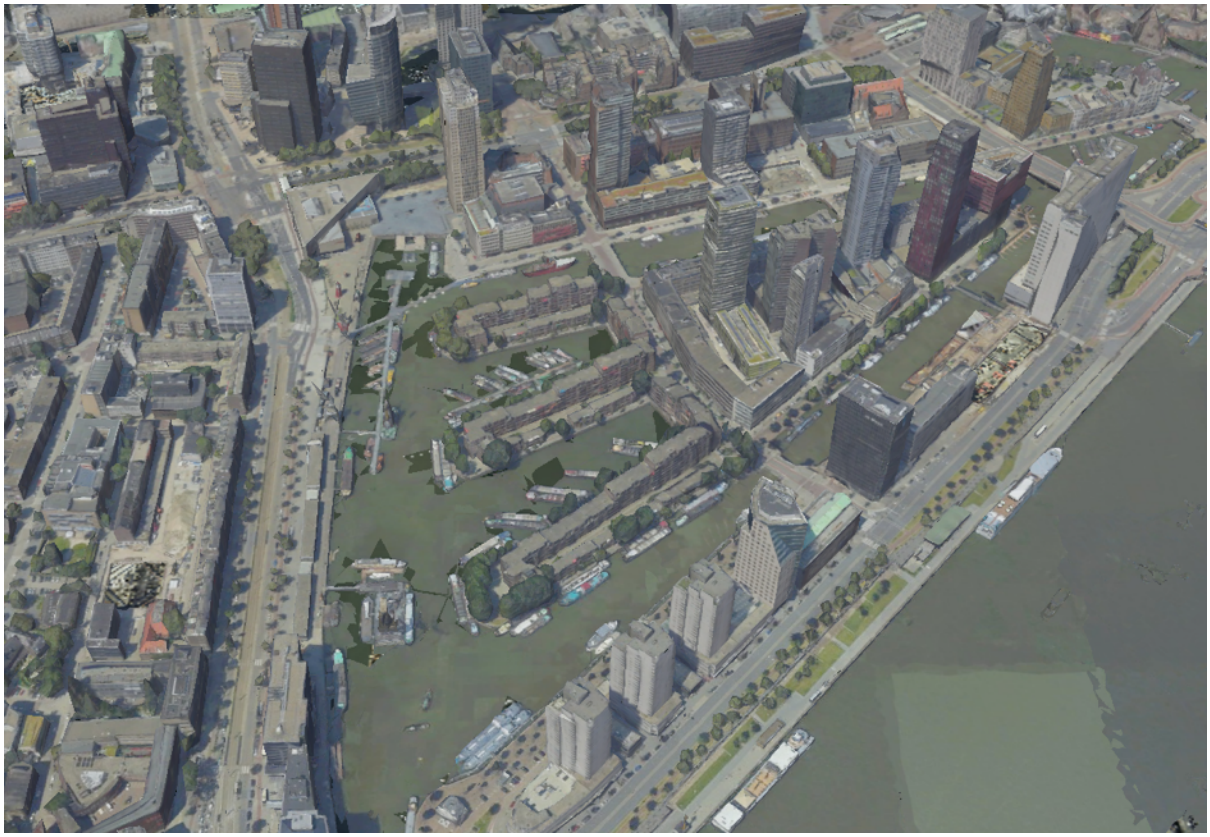


Figure 2: Screenshot of The Netherlands in 3D

3D formats for the project

Selecting the appropriate 3D file formats is essential for efficient content development, cross-platform compatibility, and long-term scalability within the CitiVerse project. Since XR tools in this project will likely rely on popular game engines like **Unreal Engine** and **Unity**, it's important to focus on file formats that are both versatile and well-supported by these platforms. The most relevant and widely adopted formats include **glTF**, **FBX**, and **OBJ**.

glTF (.gltf / .glb)

The **glTF (GL Transmission Format)** is an open standard developed by the Khronos Group and is rapidly becoming the preferred format for real-time 3D applications. glTF supports features like PBR (Physically Based Rendering) materials, scene hierarchy, animations, and embedded textures in a lightweight, efficient package.

- **In Unity**: glTF is not natively supported, but there are several high-quality open-source and commercial importers (e.g., UnityGLTF, Piglet), which allow real-time importing and runtime rendering of glTF models.
- **In Unreal Engine**: Epic Games officially supports glTF through plugins like glTF for UE. It allows both import and runtime use, making it suitable for dynamic asset streaming.

File Size Challenges and the Need for Scalable 3D Formats

While **glTF (Graphics Library Transmission Format, .glb)** is an efficient and open format suitable for XR environments, its performance depends heavily on how assets are structured and optimized. In practical use, even optimized `.glb` files can become quite large, often exceeding **700MB** for a moderately complex urban scene. This can lead to serious issues in XR applications, including long load times, frame rate drops, memory overflows, and device overheating, particularly on mobile and standalone headsets.

Large `.glb` files are essentially monolithic packages. When used in real-time applications, they must often be fully downloaded and parsed before any part of the scene can be displayed. This makes them **inefficient for streaming or incremental loading**, and unsuitable for very large environments like full cities or districts represented in the CitiVerse platform. Once loaded, the entire geometry remains in memory, regardless of whether all of it is visible to the user, putting unnecessary strain on system resources.

To address this, the CitiVerse project should consider adopting **3D Tiles**, an open specification developed by Cesium and supported by the OGC (Open Geospatial Consortium) for representing large-scale 3D environments.

3D Tiles

3D Tiles are designed specifically for **streaming large-scale 3D datasets** such as entire city models, terrain, buildings, and infrastructure. Unlike glTF files, 3D Tiles structures content hierarchically and spatially, allowing engines to load only the tiles that are visible and relevant to the user's current view and position.

Key advantages include:

- **Efficient Streaming**: Tiles are loaded on demand, drastically reducing initial loading times and allowing experiences to start quickly.
- **Memory Optimization**: Only necessary geometry is loaded into memory at any given time, which is critical for XR devices with limited resources.
- **Level of Detail (LOD)**: 3D Tiles support LOD out-of-the-box, which means high-detail versions of models are shown only when the user is close, while lower-detail tiles are used at a distance.
- **Integration with Geospatial Data**: 3D Tiles can align easily with city-scale geospatial data (e.g., CityGML, BIM, OSM), aligning well with the needs of CitiVerse and Local Digital Twins.

FBX (.fbx)

The **FBX (Filmbox)** format is widely used in 3D production pipelines due to its rich feature set, supporting skeletal rigs, animations, blend shapes, camera data, and more. Although it is a proprietary format owned by Autodesk, it remains a standard in most professional 3D tools and engines.

- **In Unity**: FBX is fully supported and serves as the default interchange format for animations and rigged characters. Unity includes built-in FBX importers that handle meshes, animations, and even basic materials.

- **In Unreal Engine**: FBX is also the primary import format. It supports static meshes, skeletal meshes, animations, and LODs. Unreal provides robust tools for importing and previewing FBX content, making it a cornerstone of its asset pipeline.

OBJ (.obj)

The **OBJ format** is a simple, text-based geometry format that supports static meshes, UVs, and basic material data. It does not support animation, hierarchy, or advanced shading, making it ideal for simple, static assets or early prototyping.

- **In Unity and Unreal Engine**: OBJ files are supported natively and can be imported without plugins. However, they are usually replaced by FBX or glTF in production environments due to their limitations.

When to use which 3D format

Format	Best Use Cases	Strengths	Limitations	When to Use
glTF (.glb/.gltf)	Real-time rendering, runtime loading, cross-platform XR apps	Open standard, compact, supports PBR, embedded assets	Not optimal for large scenes or streaming, especially on limited hardware	When scenes are mid-sized, static, and need efficient runtime performance. Ideal for mobile XR.
FBX (.fbx)	Animation-heavy assets, rigged characters, complex pipelines	Rich feature set (animations, rigs), wide tool support	Proprietary, heavier files, less efficient for web streaming	When working with animation pipelines or importing assets from 3D authoring tools into Unity/Unreal
OBJ (.obj)	Simple static geometry, early-stage prototyping	Simple, lightweight, widely supported	No animation, no hierarchy, no PBR support	Use for basic geometry imports, quick prototyping, or static, background elements
3D Tiles	Large-scale city models, streamed environments, LOD support	Excellent for streaming, LOD built-in, geospatial alignment	More complex setup, not always supported natively in engines	When dealing with full city or district environments that must stream efficiently and adapt to user perspective

Table 1: When to use which 3D format

Requirements for 3D Environments

Visual and Artistic Requirements

High-quality visual design plays a key role in XR tools, as it directly affects how users experience and interact with digital environments. In the context of the CitiVerse project, 3D environments must not only support technical functionality but also enhance immersion, engagement, and understanding.

Creating immersive and believable environments depends heavily on visual detail and quality. Elements such as lighting, shadows, reflections, and materials should be carefully applied to enhance realism without overwhelming the user. Ambient and contextual storytelling through the environment, such as landmarks, signage, or animation, can help users understand where they are and what is happening around them without requiring additional instructions.

At the same time, visual quality must be balanced with performance. XR tools are often used on devices with limited computing power, including standalone VR headsets or mobile AR platforms. Therefore, the environment must be optimized. The methods and best practices for achieving this are discussed further in the Technical Requirements section.

Finally, close collaboration with other tasks in WP3 is essential. Visual design must be informed by user experience guidelines and coordinated with technical integration requirements and data APIs. In particular, the environments must visually integrate data coming from Local Digital Twins, such as static 3D models and real-time sensor data. This cooperation ensures that the environment is both aesthetically rich and functionally useful.

Technical Requirements

1. Performance Optimization

To deliver seamless and immersive experiences in XR, 3D environments must be optimized for high performance. Prior project deliverables on performance budgets and device targets already cover much of this. A key priority is maintaining a stable frame rate of aa minimum of 90 frames per second (FPS), ideally 120 FPS as lower frame rates can lead to motion sickness and reduce user comfort (Wang et al., 2023). Performance becomes even more critical in mobile VR and standalone AR devices, where hardware resources are limited compared to desktop systems. Nonetheless, augmented reality does not require the same high-performance standards as VR, as the user is still grounded in the real world and is therefore not subject to motion sickness.

	Standalone VR	PCVR	Mobile AR
Target FPS	90 – 120 FPS	120 FPS	60 FPS
Recommended specifications	Snapdragon XR2 Gen2 or XR2+ Gen2, Apple M-series	GPU: Nvidia RTX 3070/4060 or higher, AMD RX 6600XT or higher CPU: 12th gen Intel Core i7 or higher, 7Th gen AMD Ryzen 7 or higher RAM: 32 GB	Apple A16 Bionic or newer / Snapdragon 8 Gen 2 or newer
Devices	Meta Quest 3, Meta Quest 3S, Apple Vision Pro, Pico 4 Ultra	PCVR headsets with support for 120 Hz (Valve Index, HP Reverb G2, PlayStation VR2, standalone headsets…)	iPhone 13 or later (Pro models recommended for LiDAR-scanner) Recent Android flagships or mid-range devices (e.g. Galaxy S22+)
Performance considerations	Optimize shaders and polygon sizes, optimize level of detail, use foveated rendering when possible	Use foveated rendering when possible, use of frame extrapolation through DLSS / FSR	Optimize ARKit/ARCore performance

Table 2: Target FPS, reference device specs, and performance tips by platform

One of the most effective optimization techniques is reducing the polygon count of 3D models and minimizing texture sizes. This reduces the computational load during rendering, making the application faster and smoother. Merging meshes is another useful strategy, as it decreases the number of draws calls the system needs to handle especially important in complex environments with many objects. Additionally, baked lighting can replace expensive real-time lighting calculations, significantly improving runtime performance without sacrificing visual quality. Pre-processing and caching frequently used data can also limit the need for real-time computation, which improves loading times and runtime responsiveness.

Both Google's and Meta's XR performance guidelines emphasize these points. Google recommends limiting overdraw and reducing the complexity of fragment shaders, while Meta advises keeping CPU and GPU-frame times well below the 16ms mark to maintain responsiveness. Developers are also encouraged to adopt asynchronous loading for heavy assets and prioritize occlusion culling to avoid rendering unseen geometry (Zayne, 2023; Google, 2025; Meta, 2025). Together, these practices ensure that XR experiences remain smooth and comfortable, even on resource-constrained devices.

2. Scalability and Reusability

The design of 3D environments in CitiVerse must support future scaling and evolving project requirements. Scalability ensures that environments can grow in complexity or size as needed, without requiring a full redesign. This is achieved through a modular design approach, breaking environments down into interchangeable components that can be independently updated, reused, or extended. For instance, instead of designing an entire street scene as a single static model, it can be assembled from individual modules such as sidewalk segments, lamp posts, traffic signs, building facades, and plant boxes. These components can be rearranged or replaced without affecting the rest of the environment.

Reusability also plays an important role in minimizing development time and costs. Assets such as textures, models, and shaders should be created in a way that allows them to be reused across multiple use cases and scenarios. For instance, Textures like brick walls, concrete, or asphalt can be applied across many building facades or ground surfaces. By designing with modularity and reuse in mind, the environments remain adaptable, maintainable, and cost-efficient over the course of the project.

3. Cross-Platform Compatibility

CitiVerse 3D environments must run reliably across different XR platforms, including desktop-based VR headsets, standalone devices, and AR-enabled mobile phones. This requires strict adherence to platform-specific performance guidelines and technical specifications. Platforms such as Meta's Horizon OS and Google's Android-based XR tools offer detailed performance optimization checklists, which developers are encouraged to follow.

To ensure compatibility, assets should be exported in standardized formats like glTF, FBX, or OBJ, which are widely supported across platforms. In addition, environments must be tested on a range of devices to identify issues such as resolution differences, controller input variations, or visual inconsistencies. This thorough testing helps prevent usability issues and ensures that all users, regardless of their device, can access the XR experience as intended.

4. Integration with Local Digital Twins (LDTs)

XR environments must be built with seamless integration into Local Digital Twin (LDT) systems in mind. These digital twins provide real-time and static data that XR tools rely on to create meaningful and context-aware experiences. For example, a 3D model of a city street may need to display current traffic or sensor data, which must be fetched and visualized accurately in the XR interface.

To support this, environments must align with the data formats and structural logic used in LDT architectures. They should be capable of ingesting real-time data streams such as traffic, air quality, or sound impact and visually updating elements of the environment accordingly. Additionally, care must be taken to maintain data consistency, particularly when XR environments are used in critical decision-making scenarios or simulations. Synchronization mechanisms and update checks help avoid discrepancies between what the digital twin reports and what users see in XR.

In summary, the technical foundation of CitiVerse's 3D environments must balance performance, compatibility, and interoperability. By applying industry best practices from Google, Meta, and other XR leaders, and aligning closely with the data requirements of Local Digital Twins, the project ensures that its XR tools are efficient, scalable, and ready for real-world smart city use cases.

Efficiency and optimization for 3D assets and models

To make sure most parties can use the applications; we need to use optimalisation. This is mostly because of the limited hardware of the users. If it's not done enough the application might result in an overall bad experience because of performance. For example, stuttering of motion can lead for example to motion sickness for people.

We have a few ways to lower the cost of 3D models:

- Reducing polycount & retopology
- Light/texture/geometry baking
- Lower texture quality
- Adjust level of detail (LOD)

Reducing polycount & retopology

When using unoptimized 3D models like BIM models, 3D scans and unexperienced designers we must look out for a (too) high poly count. A problem that can occur is that an object has multiple faces merged, which makes it hard to make a good-looking result. Merging points together doesn't always solve the problem and can lead to having to remake the object. A way to solve the problem is using retopology tools. Before retopology, decimation and modifiers can be used to quickly lower polycount and prepare the mesh for a cleaner result. If done correctly we can use and alter the object in a shorter time period and alterations of the same object is more within reach. This is only about the poly count and for making better 3D objects. In order to make it look better we have to alter the material/texture of the object.

Step 1) Set targets: Define a clear poly budget for assets and LODs.

Step 2) Inspect the mesh: Check for merged faces, non-manifold geometry, and errors.

Step 3) Strip junk: Remove hidden faces, duplicates, and unnecessary details.

Step 4) Retopology: Rebuild clean topology with quads and proper edge flow.

Step 5) UVs: Unwrap efficiently to prepare for texture baking.

Step 6) Validate & optimize: Fix normals, transforms, pivots, and scaling.

Step 7) Export & test: Export with compression and verify performance on device.

The steps you listed will ensure a balanced trade-off between visual fidelity and rendering performance. Many workflows (Blender, Maya, 3ds Max) recommend this sequence, sometimes with slight variations depending on the tool or final use case.

Light/texture/geometry baking

Once we have a more optimized version of the model, we might have some trouble how it looks. This might be because of the stretching and difference of the unwrap during the altering of the object. To fix this part, we unwrap the lower poly version. Once done, we can start by pushing the details of the higher poly version to the lower variant. The plus of doing this is to get an optimized prop that can be used in applications which can result in less stuttering and performance issues in lower end hardware. However, it does require more manual labor to get this done. You do have room for modifications, although that will result in more time per asset that needs to be made the same way.

Step 1) Prepare the optimized model: Ensure the low-poly model is finalized.

Step 2) UV unwrap the mesh: Redo or adjust the UV mapping on the optimized version.

Step 3) Set up materials and image textures: Create unique materials for each object.

Step 4) Configure baking settings: Set render engine & choose desired bake type.

Step 5) Execute the bake: With all settings checked, bake the desired maps.

Step 6) Export & test: Review baked textures for artifacts, edge bleeding, or mismatches.

Baking transfers visual detail to the low-poly mesh, keeping performance high without sacrificing appearance. This process ensures optimized props look as expected in the application and run efficiently even on lower-end hardware.

Lower texture quality

Textures are often very important to make it prettier and more believable. However, textures can have big sizes and can cost allot of performance. Low quality textures can make it look blurry and unreadable. Per 3D object there needs to be a check how big the object is compared to the environment and how important it is for the overall application. This process can help with the overall performance, but you need to make sure both the 3D model and texture is adjusted for the right reasons.

Step 1) Evaluate importance: Check how visible/important each object is in the scene.

Step 2) Resize textures: Downscale large textures so it matches the object's screen size.

Step 3) Compress formats: Use efficient formats (e.g., WebP) for smaller files.

Step 4) Channel packing: Combine grayscale maps into one texture.

Step 5) Remove unused maps (e.g., empty normal/specular maps).

Step 6) Export optimized assets: Save the textures with compression enabled

Step 7) Test visual quality: Compare before/after to ensure readability isn't lost.

Combine multiple techniques for the best results adjust bit depth and resolution only when possible since quality remains important for main elements and interfaces, and regularly evaluate visual quality by running test builds on the target hardware.

Adjust level of detail (LOD)

An option in game engines is to use the level of details of 3D objects to make it less demanding and extensive once it hits a certain distance. Like real life, distanced objects are harder to read and blurrier. Most often are the LODs made beforehand in one of the modelling software.

Step 1) Plan LOD tiers: Define LOD0–LOD2/3 per asset

Step 2) Build LOD meshes: Create simplified versions in your modeling tool (DCC)

Step 3) Unify materials/UVs: Reuse UVs and materials across LODs

Step 4) Prepare collisions: Use simpler colliders or disable physics on far LODs

Step 5) Import & group: Set up LOD Groups/HLOD in the engine with consistent naming

Step 6) Configure thresholds: Use screen-relative size or distance thresholds

Step 7) Cull & stream: Pair LODs with frustum/occlusion culling and asset streaming

Step 8) Test and tune on device

Many modern programs and render engines do automate parts or most of these LOD steps. However, manual tuning and validation remain important to achieve the best visual quality and performance balance for each project.

Design Framework

These steps outline a simple, practical framework for designing intuitive XR UI/UX design framework. We start by aligning on goals, users, and key tasks, then shape principles, interactions, and visuals that work across devices. Keep it light, test early, and let real user feedback guide what we build next.

Step 1) Agree on goals

The first step to build your UX/UI design framework is to agree on the goals. Before we design anything, we need a shared, simple set of goals. In one page, write down who we're building for, the top three things they need to do, and how we'll know it works. Keep it concrete: which devices we target (desktop VR, headset, phone), where the app will be used (office, street), and the few numbers that define success (e.g., complete a building lookup in under 30 seconds, keep 60+ FPS, zero nausea reports). If a feature doesn't support one of these goals, it waits. This keeps the product focused and the experience calm and clear for users.

Example: for 1 use case, we are focusing on traffic visualization. Our users should be able to open the city scene, pick a road, and see live congestion and a short forecast in under 20 seconds. On a phone in AR or a headset in VR. If a feature doesn't support these goals, it waits. This keeps the experience focused, calm, and useful.

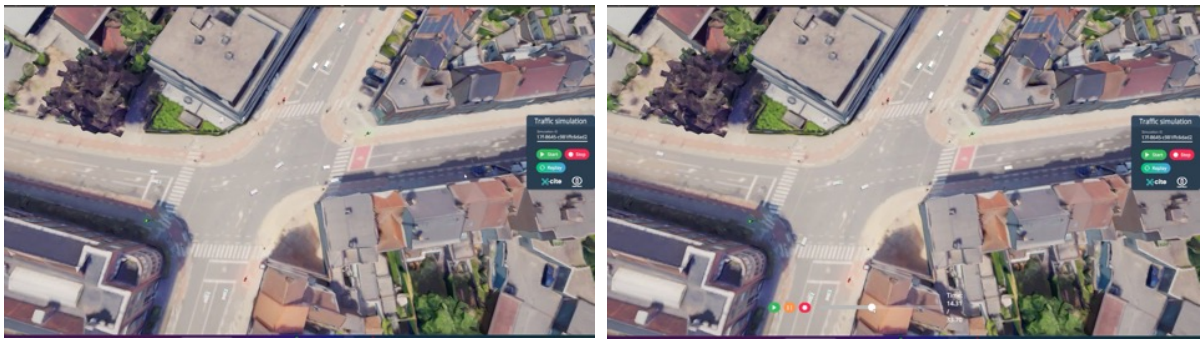
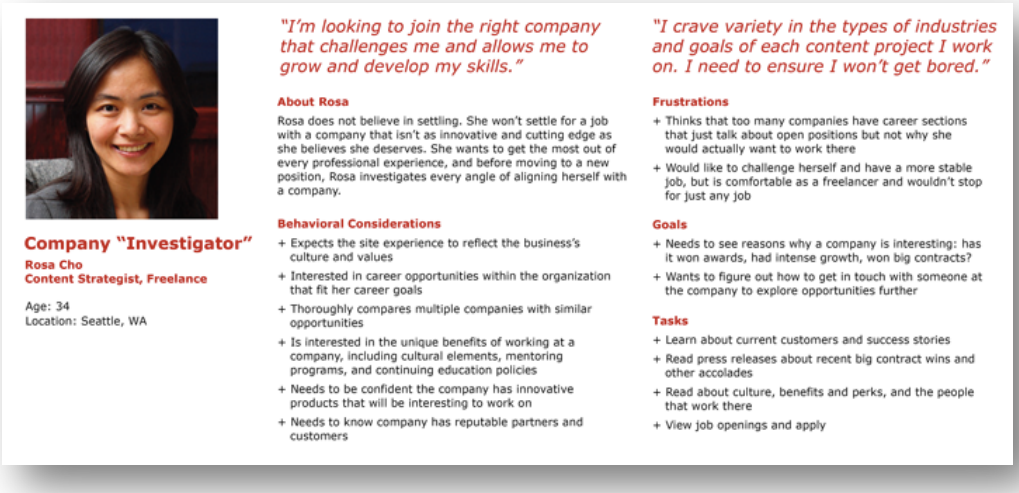


Figure 3: First demo of the 3D traffic viewer tool for the Flemish use case (screenshot)

Step 2) Define 3 core personas

Create three clear personas that feel like real people. Give each a name and a photo and a one sentence goal line. Describe where they use the tool and which devices they carry and the top tasks they must finish. Add any constraints or accessibility needs. Add simple success criteria such as find a congested road and view a short forecast in under twenty seconds. Keep each persona on one page and pin them up so the team can point to them when choices are unclear.



Company "Investigator"
Rosa Cho
Content Strategist, Freelance

Age: 34
Location: Seattle, WA

"I'm looking to join the right company that challenges me and allows me to grow and develop my skills."

About Rosa

Rosa does not believe in settling. She won't settle for a job with a company that isn't as innovative and cutting edge as she believes she deserves. She wants to get the most out of every professional experience, and before moving to a new position, Rosa investigates every angle of aligning herself with a company.

Behavioral Considerations

+ Expects the site experience to reflect the business's culture and values
+ Interested in career opportunities within the organization that fit her career goals
+ Thoroughly compares multiple companies with similar opportunities
+ Is interested in the unique benefits of working at a company, including cultural elements, mentoring programs, and continuing education policies
+ Needs to be confident the company has innovative products that will be interesting to work on
+ Needs to know company has reputable partners and customers

"I crave variety in the types of industries and goals of each content project I work on. I need to ensure I won't get bored."

Frustrations

+ Thinks that too many companies have career sections that just talk about open positions but not why she would actually want to work there
+ Would like to challenge herself and have a more stable job, but is comfortable as a freelancer and wouldn't stop for just any job

Goals

+ Needs to see reasons why a company is interesting: has it won awards, had intense growth, won big contracts?
+ Wants to figure out how to get in touch with someone at the company to explore opportunities further

Tasks

+ Learn about current customers and success stories
+ Read press releases about recent big contract wins and other accolades
+ Read about culture, benefits and perks, and the people that work there
+ View job openings and apply

Figure 4: Persona example from Nielsen Norman Group | Harley, A. (2015, February 16). Personas make users memorable for product team members. Nielsen Norman Group. https://www.nngroup.com/articles/persona/

Step 3) Set design principles

The UI and UX needs to follow a few simple rules that fit the CitiVerse vision. We use design principles to keep the team aligned on clear, user-centric decisions so every screen stays consistent, accessible, fast, and fit for the project context. Here are four design principles aligned with the X-CITE vision.

People first: Design for citizens, planners, communities, government and field staff. Use clear language, readable UI, and options for different abilities so everyone can take part. x-CITE aims to make cities smarter, greener, and more inclusive with citizens at the heart

Open and interoperable: Build with open standards and the Local Digital Twin toolbox so tools plug into many city platforms without rework. This matches the EU push toward a shared CitiVerse and networked LDTs.

Fast on real devices: Keep frames smooth and loading light. Stream only what is needed, use simple layouts, and make key actions quick so the experience works in offices, on the street, and on headsets or (older) phones.

Co-create and build trust: Involve users early, show where data comes from, and protect privacy by default. The project promotes human-centred, ethical use of XR in cities.

Step 4) Choose the right interaction model

Choosing the right interaction model matters because what feels natural on a keyboard and mouse is very different from touchscreen or hand controllers in a headset. Choosing device-appropriate patterns makes tasks faster, reduces errors and discomfort, and keeps the experience accessible and trustworthy.

In a **3D web apps** for regular users, navigation should feel familiar: move/drag to pan, use the mouse wheel to zoom, and hold the right button to orbit the view. Arrow keys or WASD (qwerty) can control movement for those who prefer the keyboard. Keep layer toggles and the legend in the top-right, and make sure every control is reachable by keyboard and works with screen readers. Aim for a calm, uncluttered layout that stays fast and easy to read.

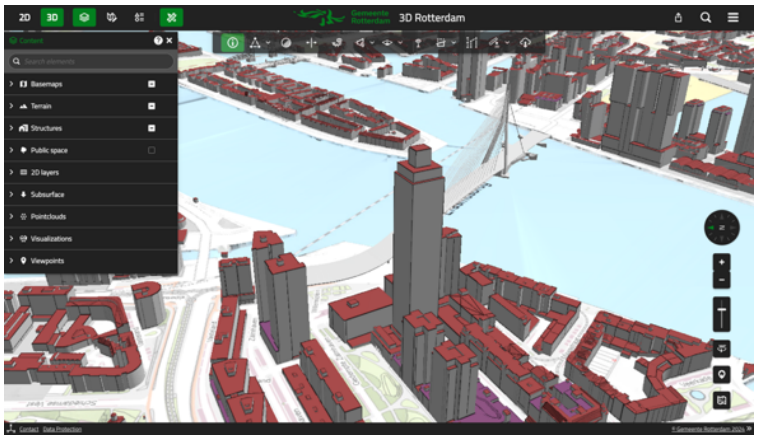


Figure 5: 3D Rotterdam works with a similar interaction model

In a simple **AR app**, people move the phone to explore and tap to place or select something in the scene. Use familiar gestures: pinch to scale, two-finger twist to rotate. Keep a small world-locked info card near the object with one clear action button and an easy recenter. Make camera permission and capture status obvious and avoid face capture by default. Keep labels out of the way, readable in sunlight, and tap targets large for one-hand use. Stay light and fast so it feels instant, like popular AR games or built-in product viewers.



Figure 6: Pokemon GO uses the camera feed a main backdrop. The Pokémon GO team. (2017, December 20). AR+ is now available in Pokémon GO! Pokémon GO News. https://pokemongo.com/post/arplus



Figure 7: Quick Look AR lets you access AR experiences directly from the browser. Apple Inc. (n.d.). Augmented Reality. Apple (Belgium). https://www.apple.com/benl/augmented-reality/

In a **VR app**, use standard controls so anyone can pick it up: teleport to move and snap-turn for comfort, point with a laser and press the trigger to select, and show a floating panel at a comfortable distance when details are needed. Let users two-hand grab to scale or reposition the scene and release to recenter, keep a wrist menu for quick actions, and use a large radial control for adjustments with gentle haptic ticks for feedback. Prioritize comfort, clarity, and speed: steady high frame rate, crisp readable text and big target.

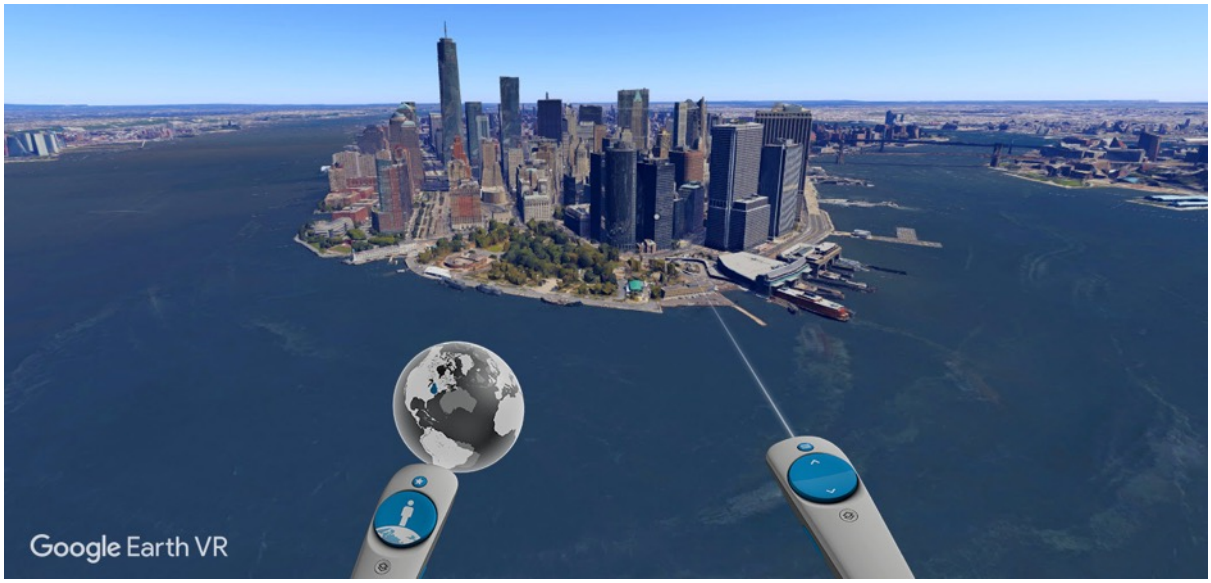


Figure 8: Google Earth VR is a free VR experience. Google. (2016, November 16). Google Earth VR. Steam. https://store.steampowered.com/app/348250/Google_Earth_VR/

Step 5) Organize spatial UI layout

Keep main panels steady and easy to read. In 3D, place them at a comfortable distance. On 3D screens, keep them in clear, consistent spots. Keep controls in the main viewing area and remove anything that clutters the view. Link labels to the correct object with a clean line and space them so they never overlap.

Wireframe early. Start with quick sketches, try rough grey boxes, make a simple clickable mockup, and only polish the design after you test what works.

Step 6) Choose your visual language

Create a small visual language that works the same in 3D, AR, and VR. Pick a few colors: base, surface, overlay, plus clear colors for success, warning, and alert, with strong contrast. Set three text sizes: small, body, and heading, and make them easy to read briefly. Use one clean line icon set across all screens. Keep materials soft and semi-matte and avoid busy textures behind text and buttons. Put it all on one page with the colors, text sizes, icons, spacing, and simple motion rules, and keep a small material library you can reuse in every app.



Figure 9: Different visual iterations for the Rotterdam use case. Working book Living Magazine | Krill o.r.c.a. // Harmen van de Wal

Step 7) Create your component library

Build a small component library that you can reuse across all apps. Use clear buttons, toggles, drop down menus and sliders. Add search with suggestions, a layer manager with easy on and off, a legend that matches colors and symbols, callouts that pin to the right spot with a clean line, and an anchor placer that snaps to surfaces. For each component, write one short do and don't and a note on when to use it, so everyone builds the same way. Keep targets large, text readable, and the app fast. For inspiration, look at existing UI sets like Meta's Horizon OS patterns and adapt only what fits your use case.

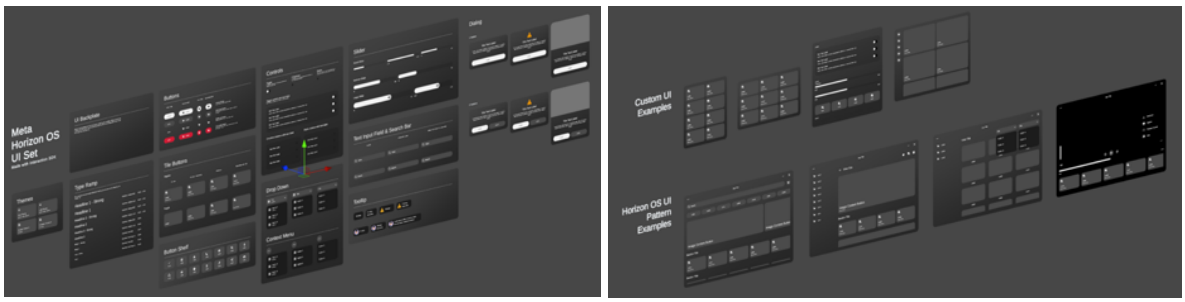


Figure 10: Several Meta Horizon OS UI Set components. Meta. (2025, March 18). UI Set. Meta Developers. https://developers.meta.com/horizon/documentation/unity/unity-isdk-uiset/

Step 8) Think about accessibility and comfort

Set helpful defaults so everyone can use the app. Offer seated and standing modes. In VR, use snap turn and a gentle vignette to reduce motion sickness. Let people scale text and UI, turn on high contrast, and pick a color-blind-safe palette. Support one-handed use. Add captions or subtitles for audio. Limit fast motion and avoid flashing or flicker.

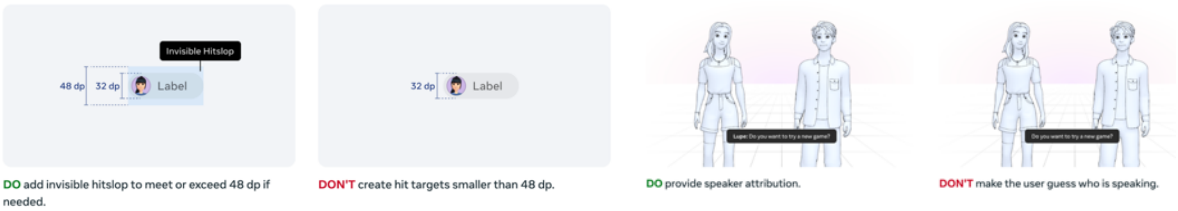


Figure 11: Meta gives useful tips for more inclusive VR apps. Meta. (2025, September 11). Accessibility. Meta Developers. https://developers.meta.com/horizon/design/accessibility

Step 9) User Acceptance Testing

Finish with a short, real-world user acceptance test to confirm the product meets needs and earns approval from real users: residents, city government, and field staff. Run a small pilot on the actual devices, connect to the Local Digital Twin, and verify that data, layers, and timestamps align. Check privacy and accessibility settings, measure basic performance, and collect clear accept or reject feedback for each user goal. Record what you learned and update the design framework so the next app starts stronger.

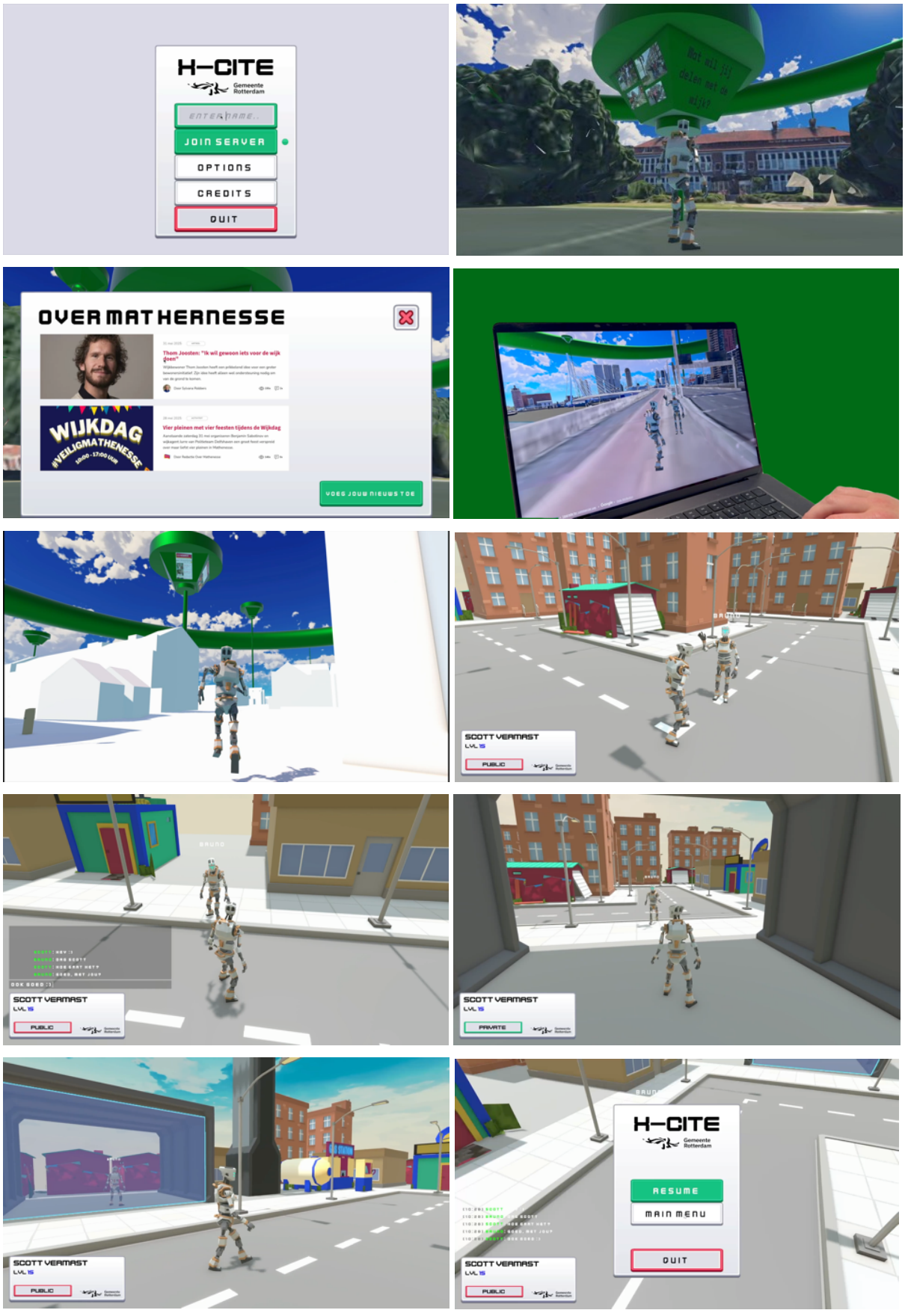


Figure 12: Proposed mockups for “Rotterdam use case - iteration 0”. Implementation of login systems, menu’s, chatbot, privacy zones & information points.

Propose a secure and scalable dataservice API

Secure API design is crucial to prevent API attacks and ensure data integrity. This involves:

- Expose only the data and actions that are truly needed.
- Make sure each endpoint behaves correctly for its context.
- Preventing protocol or infrastructure-based data leaks

Implementing strong authentication and authorization mechanisms, such as OAuth and JSON Web Tokens (JWT), is essential for securing APIs. This involves verifying the identity of users and systems interacting with the API and ensuring that authenticated users have appropriate permissions.

Example: using JWT to authenticate users can help prevent unauthorized access.

API gateways can provide an additional layer of security, including rate limiting. This involves deploying API gateways to manage API traffic and handle authentication and authorization tasks.

Example: using Amazon API Gateway can help secure APIs and prevent attacks.

Data validation and sanitization techniques are necessary to prevent common vulnerabilities, such as SQL injection and cross-site scripting (XSS). This involves validating all input received through APIs and ensuring that inputs are checked against a strict type, length, format, and range.

Example: using regular expressions to validate input data can help prevent SQL injection attacks.

Scalability and performance optimization are critical concerns for APIs, and techniques such as caching and rate limiting can be used to improve API performance. This involves optimizing API design and implementation to handle large amounts of data and traffic.

Example: using caching to store frequently accessed data can help improve API performance.

Components of a secure and scalable dataservice API

1) Interoperability-first

We prefer open, widely adopted standards, such as 3D Tiles, SensorThings and glTF 2.0, to name a few. Open standards ensure vendor-neutral integration and futureproofing. The core protocols include OGC API Features (for RESTful geospatial access with JSON/GeoJSON support), 3D Tiles for hierarchical 3D scene delivery, SensorThings API

for time-series observations with MQTT support and glTF 2.0 with Draco compression for portable 3D assets.

2) XR-ready

This includes streaming large 3D city models & assets, support for LOD, tiling, and compression. This pipeline contains automated LOD & tiling generation, progressive streaming with HTTP range requests and Brotli compression for static assets plus WebRTC for dynamic patches, GPU-friendly compression using Draco for meshes and textures. The architecture supports hierarchical level-of-detail (HLOD) for optimal rendering based on distance and importance

3) Security & privacy by design

Keep the CitiVerse simple and safe from day one. Use OAuth 2.0 and OpenID Connect for sign-in, give people only the access they need, and protect all traffic with TLS. Separate tenants in both policy and storage, encrypt data in transit and at rest, log access, and run regular security tests. Follow GDPR by limiting personal data at every step: collect only what you need, keep it only as long as necessary, and share as little as possible.

For the 3D environments, avoid storing personal data inside scene assets. Meshes, textures, and materials should be non-personal. Use derived overlays for live information and keep any raw images, depth, or biometrics on the device whenever possible. Persistent anchors should store coordinates and accuracy only, not photos. If personal data is needed, store it in the Local Digital Twin or another backend with clear purpose, legal basis, access rules, and short retention; the XR tool should fetch only the minimum required. When processing could be high risk, such as large-scale sensing, tracking, or biometrics, run a DPIA before launch and record mitigations; if risk remains high, consult the national data protection authority.

According to the European Commission, NIS2 expects multi-factor authentication, clear incident-response and business-continuity plans, management accountability, and a reporting timeline: early warning within 24 hours, a detailed notification within 72 hours, and a final report within one month or a progress report if the incident is still ongoing. These duties apply to the entire CitiVerse stack**:** the data service API, the 3D environments, client apps, and the backends they connect to.

The core NIS2 rules are EU-wide, but each country sets its own law, timelines, authorities, and procedures. Belgium has already transposed NIS2 and designated the Centre for Cybersecurity Belgium (CCB) as the national authority, so Belgian entities follow the national regime. The Netherlands is still finalising its implementing law (Cyberbeveiligingswet). Finland has implemented NIS2 via the Cybersecurity Act (124/2025), in force since 8 April 2025. In practice, design your APIs and 3D environments with these controls in mind: strong authentication, least-privilege access, secure data flows, auditing, and incident readiness across all components.

Details of where each data type lives and how flows are secured will be specified in the architecture framework deliverable that describes the LDT, XR clients, and APIs in depth.

4) Separation of concerns

Separation of concerns means dividing an API into clear parts. Like handling user access, checking data, running core logic, and storing information. Each part works on its own: security checks don't slow down data processing, and storage systems can scale up separately. This makes the API easier to update, keeps it safer by isolating sensitive tasks, and lets it grow smoothly when more users connect.

5) Cloud-native scalability

Cloud-native data APIs use lightweight containers and microservices to grow or shrink automatically based on demand. Orchestration tools (like Kubernetes) monitor performance and add or remove service instances in real time. Security is built in with mutual TLS encryption and strict access controls at every service boundary. Data is stored in distributed, replicated databases that shard across regions for low latency. This approach ensures APIs stay fast, reliable, and secure, even as traffic spikes unpredictably.

What types of data the does the API need to handle?

Data Types	Formats
Static 3D city models	CityGML, CityJSON, i3dm, b3dm, BIM/IFC derivatives, photogrammetry, point clouds: exposed as 3D Tiles for XR streaming
XR assets	glTF 2.0 scenes/models (PBR), materials, textures, animations. Delivery variants: full-fidelity, LOD, platform-targeted encodings (e.g., KTX2 + meshopt/Draco). Scene compositions, prefabs, shaders (metadata references), avatars, annotations.
Dynamic sensor & event data	IoT observables (environment, mobility, crowd, energy), device metadata, events/alerts, timeseries.
Geospatial features & rasters	Buildings/roads/POIs/landuse (vector). Terrain/imagery/coverages (raster) as tiles. Queryable features for selection/anchor placement. GeoJSON
Catalog & governance metadata	Datasets, services, licenses, lineage, usage restrictions. Discovery and cross-portal syncing.

XR scene state	Saved viewpoints, bookmarks, annotations, collaboration state, per-user layers (subject to privacy rules).
Access & policy artifacts	Identities/roles (opaque to clients), attribute grants, dataset policies, consent/retention tags, audit events.

Table 3: API data types

Bibliography

European Commission. (2025, July 1). *NIS2 Directive*. Digital Strategy. https://digital-strategy.ec.europa.eu/en/policies/nis2-directive

Google. (2025). VR Performance best practices. Google for Developers. https://developers.google.com/vr/develop/best-practices/perf-best-practices

Harley, A. (2015, February 16). Personas make users memorable for product team members. Nielsen Norman Group. https://www.nngroup.com/articles/persona

Meta. (2025). *Guidelines for VR Performance Optimization*. Meta Developers. https://developers.meta.com/horizon/documentation/native/pc/dg-performance-guidelines/

Meta. (2025, March 18). UI Set. Meta Developers. https://developers.meta.com/horizon/documentation/unity/unity-isdk-uiset/

OWASP Foundation. (2023). *OWASP API Security Top 10*. https://owasp.org/API-Security/editions/2023/en/0x00-header/

Rosala, M. (2020, August 2). *Crafting product-specific design principles to support better decision making*. Nielsen Norman Group. https://www.nngroup.com/articles/design-principles/

Wang, J., Shi, R., Zheng, W., Xie, W., Kao, D., & Liang, H. (2023). Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality. *IEEE Transactions on Visualization and Computer Graphics*, *29*(5), 2478–2488. https://doi.org/10.1109/tvcg.2023.3247057

Znanye. (2023, June 14). *How to Handle Performance Optimization in VR Development?* https://www.linkedin.com/pulse/how-handle-performance-optimization-vr-development-znanye/