eTryOn - Virtual try-ons of garments enabling novel human fashion interactions

Project Title:	**eTryOn - Virtual try-ons of garments enabling novel human fashion interactions**
Contract No:	951908 - eTryOn
Instrument:	Innovation Action
Thematic Priority:	H2020 ICT-55-2020
Start of project:	1 October 2020
Duration:	24 months

Deliverable No: D4.1

Architecture and integration Protocol

Due date of deliverable:	31 May 2021
Actual submission date:	4 June 2021
Version:	2.3
Main Authors:	Ray Miller (Metail) Jim Downing (Metail)

Project funded by the European Community under the H2020 Programme for Research and Innovation.

Deliverable title	Deliverable Title
Deliverable number	D4.1
Deliverable version	Final
Contractual date of delivery	31 May 2021
Actual date of delivery	4 June 2021
Deliverable filename	eTryOn_D4.1_final.docx
Type of deliverable	Report
Dissemination level	PU
Number of pages	42
Workpackage	WP4
Task(s)	T4.1
Partner responsible	Metail
Author(s)	Jim Downing (Metail), Ray Miller (Metail)
Editor	Elisavet Chatzilari (CERTH)
Reviewer(s)	Tasos Papazoglou Chalikias (CERTH)

Abstract	The starting point for the architecture and technical decisions made in support of the eTryOn Project use cases and pilots.
Keywords	Architecture, integration

Copyright

Deliverable history

Version	Date	Reason	Revised by
1.0	2021-04-18	Table of Contents	Jim Downing
2.0	2021-05-18	Initial version	Ray Miller, Jim Downing
-	2021-05-20	Comments by	Tasos Papazoglou Chalikias
2.1	2021-05-25	Revised version	Jim Downing
2.2	2021-05-26	ToC rebuilt	Jim Downing
2.3	2021-06-02	Final version	Elisavet Chatzilari

List of abbreviations and Acronyms

Abbreviation	Meaning
KPI	Key Performance Indicator
DevOps	A combination of Software **Dev**elopment and IT **Op**eration**s**. DevOps is a set of practices that combines software development and IT operations. It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.
DSL	Domain Specific Language
API	Application Programming Interface. In this document it exclusively means a network interface.
AR	Augmented Reality - the composition of 3D objects into a scene viewed through a camera and display that adds virtual objects to the scene.
HTTP (S)	HyperText Transfer Protocol (Secure). HTTP is the protocol used to transfer data over the web. It is part of the Internet protocol suite and defines commands and services used for transmitting web data.
NoSQL	Database management systems that do not use the relational model or offer querying in Structured Query Language (SQL). These often offer high throughput and easier updating and reading.
VR	Virtual Reality
ADR	Architecture Decision Record
SDK	Software Development Kit. In this document this refers to a component that can be incorporated into a larger software container and runs in-process with it.

Table of Contents

Contents

1 Executive summary

The development of architecture in eTryOn is an iterative, ongoing process that aims to promote communication and reduce technical risk on the project. We particularly focus on timely decisions of technology choices and approaches, and on clarifying interfaces between components (especially those between consortium partners).

Section 2 describes the general principles, tools, patterns and practices that apply across all the systems being developed in the eTryOn project. Section 3 goes into more depth on each of the 3 systems describing both the coordinating components that each system will consist of, as well as looking at key interactions, through architecture diagrams generated from the architectural model for eTryOn. Both sections highlight where key decisions have been made, and these decisions are described in full in section 4.

Most of the contents of this document are continuously updated and maintained through the project technical documentation website at https://etryon.gitlab.io/techdocs/ .

2 Architecture Principles, Tools and Approaches

Our aim is to create an architecture with the primary goal to help the eTryOn project development to progress smoothly and to support the use case pilots towards the end of the project, rather than to anticipate the needs of a commercial implementation after the project. It follows that the initial decisions made around architecture focused on how architectural decisions would be made and communicated between consortium partners. We chose to leverage serverless cloud technologies as we have found them to be an excellent way to build systems that scale down as well as up, allowing speed, flexibility and low development overheads during system development without significant costs scaling up.

2.1 Architecture Decision Records

We have found Architecture Decision Records (ADRs) to be a useful discipline, especially in distributed teams. By recording the context and conclusion for each architecture decision into version control we seek to clarify thoughts when making the decision, and make it possible for those who come to the architecture later to understand it better and give them a better understanding of the ramifications of making changes. More details can be found in Section 4.1.

2.2 C4 Modelling

Diagrams are crucial in the communication of software architectural models. In eTryOn, rather than relying on ad-hoc diagrams from graphics tools we chose to adopt the C4 model to encode our architectural ideas and generate diagrams from the model. Following this model gives our diagrams some reasonable semantic fidelity - it will make it easier to ensure a 1-1 correspondence between the documentation and the running systems as implementation progresses. We chose to use the C4 Domain Specific Language (DSL), which means our architecture model can be usefully handled in version control systems, allowing variations to be modelled on branches, and for parallel development to be merged. More details can be found in Section 4.2.

2.3 Dev/Ops & Collaboration Platform

To simplify collaboration over shared project information and source code, and to provide a Continuous Integration / Continuous Deployment (CI/CD) technology for the project, we decided to use a code development platform based around the git Distributed Version Control System (DVCS). We favoured Gitlab; more can be found in Section 4.3.

2.4 Living Documentation

Although this document represents the state of the architectural design at a particular project milestone, from the start we regard the architecture as an evolving, iterative process. As such, our decisions, diagrams and other use case documentation are continuously made available to consortium partners through a website. We prefer website platforms that are built with the same primitives as the rest of the code and architecture artefacts: Version controlled files. As such, we chose a static website generator, preferring Hugo for speed. The Hugo site is hosted in the project GitLab organisation, and Gitlab's CI/CD facilities are used to build and republish the site on every commit. In addition, branches are published separately, giving us the ability to easily create draft / staging versions of the website for feedback from project partners before merging the changes into the main site. See also Section 4.4.

2.5 Service-Oriented Architecture & API abstractions

The project applications will use consortium partners' pre-existing services and software, or customizations of them. A service-oriented architecture makes sense as a good basis for decoupling work package contributions, and for separation of concerns in the final software both to allow partners

to continue to develop service components beyond the project, and to minimize licensing issues in substituting parts if that becomes necessary. They are also the right places to create system interface abstractions. We decided, therefore, to develop these components behind service APIs, and allow each partner to develop the implementations as they wished behind the service APIs (see Section 4.5).

2.6 Cloth Modelling Library

Creating digital garment models that run at real-time in the physics engines supported by VR applications was a crucial area of investigation, and one that we knew we would need to optimise for, and design architecture around. The best balance of quality of cloth representation and speed currently comes from models that are able to blend tighter areas of the garment to modelled through skinning, and others to have draping physics either through particle simulation or finite elements. We considered and evaluated a number of options before settling on ObiCloth running in the Unity platform (more details can be found in Section 4.8 and deliverable D5.1).

2.7 Application Development Platform, Avoiding Low-value Middleware

In recent years, the availability of rich, high-level services in public cloud offerings like Amazon Web Services and Google Cloud Platform has allowed developers to greatly increase productivity by leveraging general purpose services for common system components like databases, web application servers, messaging queues and so on. This has led to the development of SDKs and tooling that make it easier to compose those components into client applications. For the reasons described in Section 4.6), we chose Google's Firebase platform as a foundation for the development of applications in our use cases.

Historically, it has often been the practice to partner every application front end client with a middleware layer. The function of this middleware layer is often to mediate access to other network services and to provide a layer of abstraction over data structures. We have chosen not to create middleware by default, only creating middle tier components when there is a clear business logic requirement for them. This is partly enabled by the facilities in Firebase to connect front end applications directly to the general services it offers. It has the great advantage that it avoids middleware from being a blocking dependency in project development. For example, if the developers of the front-end application need to change a data structure in order to progress, they can do so by changing the schema in the data store and their own application; there is no additional dependency for a middle tier modification. Issues of back compatibility are also minimised during project development. This flexibility, in turn, allows a much more iterative style of development.

2.8 Isolating Use Cases

As we design and develop the system components directly related to the applications and use cases in eTryOn, we have the choice of whether to develop them as one large system that can support all three use cases, or as three platforms, each supporting a single use case.

	Pro	Con
Single System	● Shared use of fixed-cost components ● Provisioning, deployment and configuration can feasibly be done manually ● System would need to be fragmented and refactored heavily for tech transfer to any one audience post-project.	● CI/CD can become complex ● Provisioning, deployment and configuration can feasibly be done manually (and is therefore error-prone) ● Difficult to reason about

		usage patterns and data flows
Multiple Systems	● CI/CD simpler for each system ● Automated provisioning, deployment and configuration is more robust. ● Systems can be independently supported for each pilot. ● Systems can be transferred from the project independently. ● Can reason about usage patterns and data flows in the scope of a single use case.	● Any fixed costs are duplicated ● Provisioning, deployment and configuration must be automated

We favour the multiple systems route (Section 4.7), having found good mitigations for the difficulties:

1. Favouring “serverless” style higher level services in cloud offerings. These will typically involve relying on the cloud provider to provision compute for your application on-demand, with the advantage that they both scale down and up; they cost very little when lightly loaded, but also scale out elastically to very high capacity, supported by public cloud service offering them.
2. Using configuration management software for provisioning, configuration and deployment. We usually use Terraform for this, but have not yet selected a configuration management solution for the eTryOn applications.

Because of this decision, the architecture models shown in Section 3 can be accurately displayed per-use case.

2.9 Privacy and Data Protection

Our approach to Data Privacy and Protection is in line with EU General Data Protection Regulation (GDPR). This section highlights some particular features of the system with respect to data privacy.

2.9.1 Data encryption

By basing our applications on Google Cloud Platform services we make use of their data encryption at rest, and we maintain encryption during transit by using HTTPS services for any transfer of personal data.

2.9.2 SAR & Right To Be Forgotten Requests

In System 3 where end user activity data (analytics) is stored in order to give recommendations, we will ensure these records are stored in a logical container per customer, making Subject Access Requests and Deletion requests straightforward to respond to.

2.9.3 Consent

In System 1 all the participants are in employment contracts that cover storage of personal data, which is our basis for storing relevant personal data.

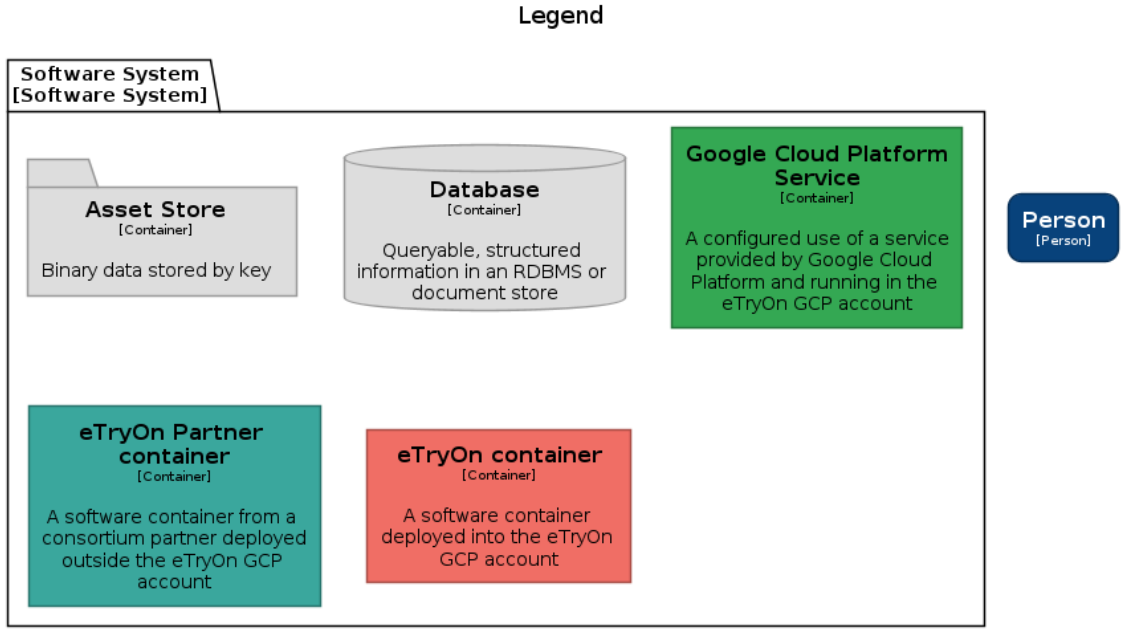
In Systems 2 and 3, we will obtain an appropriate consent from the user, and store the details of the consent collected in the Data Store.

2.9.4 Photographs and body models

We regard user photographs that are used to be particularly sensitive. Beyond the scope of eTryOn, we recognise that the body model prediction from image would ideally be performed on device for privacy reasons. Until this is possible, we have been careful to design a system that i) avoids storing photographs on more systems than entirely necessary, ii) minimises the number of data processors of this data (QuantaCorp only) and iii) gives QuantaCorp a clear event (when the results write is successful) that signals that the photographs are no longer needed and can be deleted. An example of this flow can be seen in Section 1.1.1).

3 Architecture overviews

This section describes the architectural designs to date, including the key service components needed for each use case. Sequence and interaction diagrams are provided to explain key interactions between these components in meeting use case scenarios. Some service components appear in more than one system. Where these are eTryOn components, they will be independently deployed, although they will have a common source codebase (see also Section 4.7). Where these are eTryOn partner components called through an API, it is the partner's choice whether to deploy per use case (see also Section 4.5). This legend describes the colour coding and shape keys used through this section.



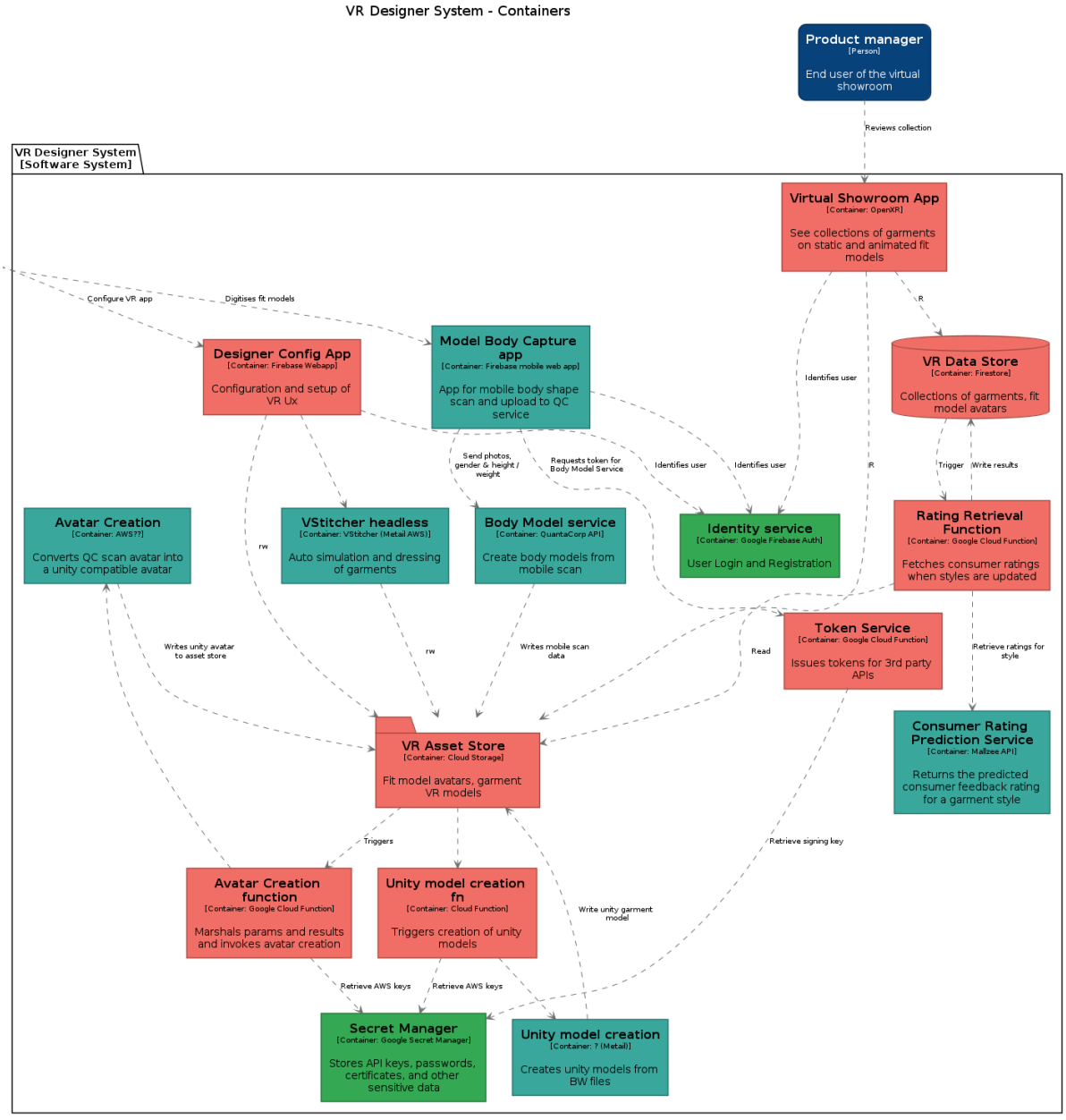
In all cases designed so far, we intend to use Google Cloud Platform (GCP) Firestore as a database. This is a document-based NoSQL database which does not enforce a schema over the data it stores, leaving that to the calling clients. This is ideal for our usage as it allows minimal blocking communications as iterative development happens (see also Section 4.6). Firestore also supports a pub/sub event system in which clients can subscribe to changes to parts of the data model, and we leverage this in our architecture, using these changes to trigger Cloud Functions that call external services.

Our Asset Stores are all based on Google Cloud Storage. As well as being scalable, resilient storage for binary data, Cloud Storage has two features that we leverage in our architecture. Firstly, it also has a pub/sub event system built in, which we use to trigger the next steps in sequential processes. This has scalability and system reliability benefits when processing steps are long-running, e.g. writing large files or calling computationally expensive external processes. Secondly, allows security through cryptographic signing of URLs to access existing data files and also to write to. We make extensive use of this when calling to external systems, any parameters to services that are at all large will be passed as pre-signed URLs, and external services write results directly to Cloud Storage through another presigned URL (rather like an "Out parameter" in C#) rather than attempting to send large HTTP requests / responses.

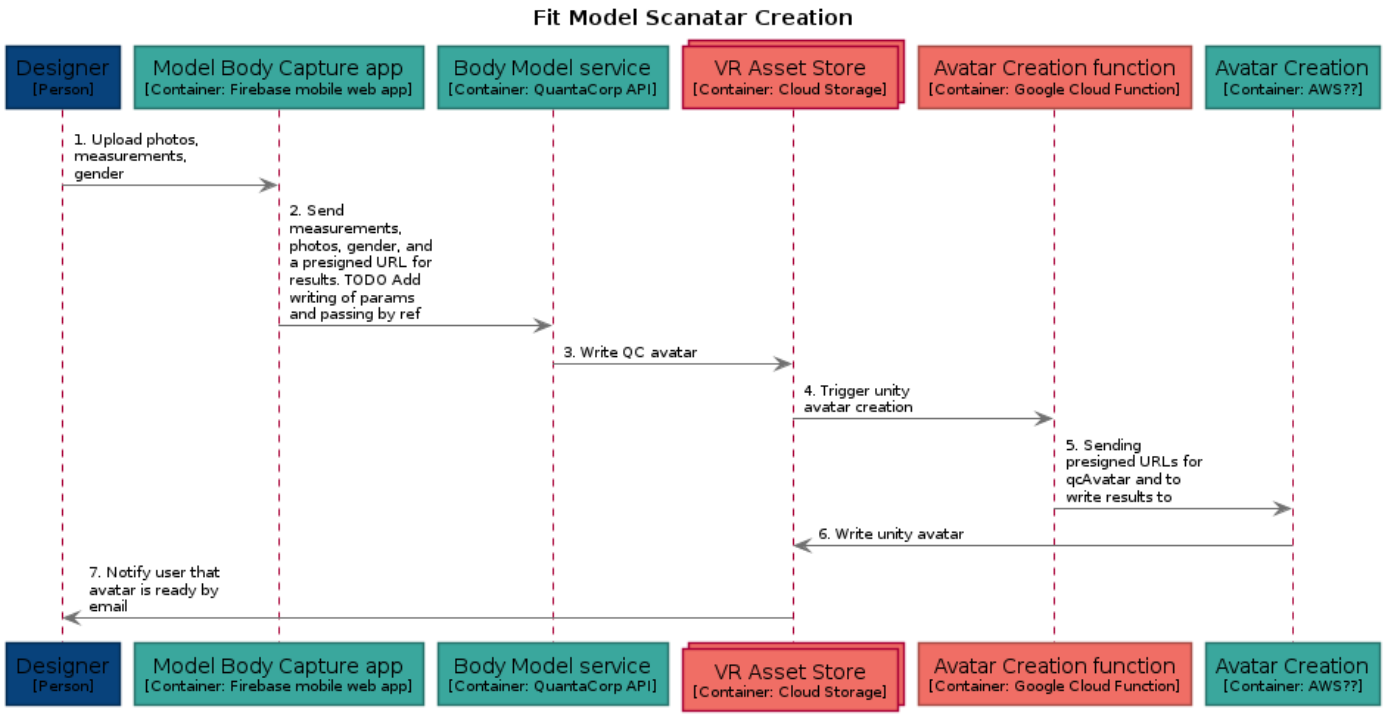
3.1 System 1 - VR Designer System

The VR Designer System contains 3 front-end applications with a number of supporting service elements. The Model Body Capture app is a mobile web application used by the Designer to create avatars for the brand's fit model. These appear in the Designer Config App (a web application), which is used to manage these avatars and to create collections of digital garments to show to the Product Manager, who will experience them through the immersive Virtual Showroom App. We plan to show garments both as view-only animations created by physics-based simulation in VStitcher (see Section 4.13) and as interactive garments. The choice of technology for all the interactive garment models in eTryOn is summarised in Section 4.8.

The choice of technology and application format for the user-facing applications is covered in Sections 4.6 and 4.9.



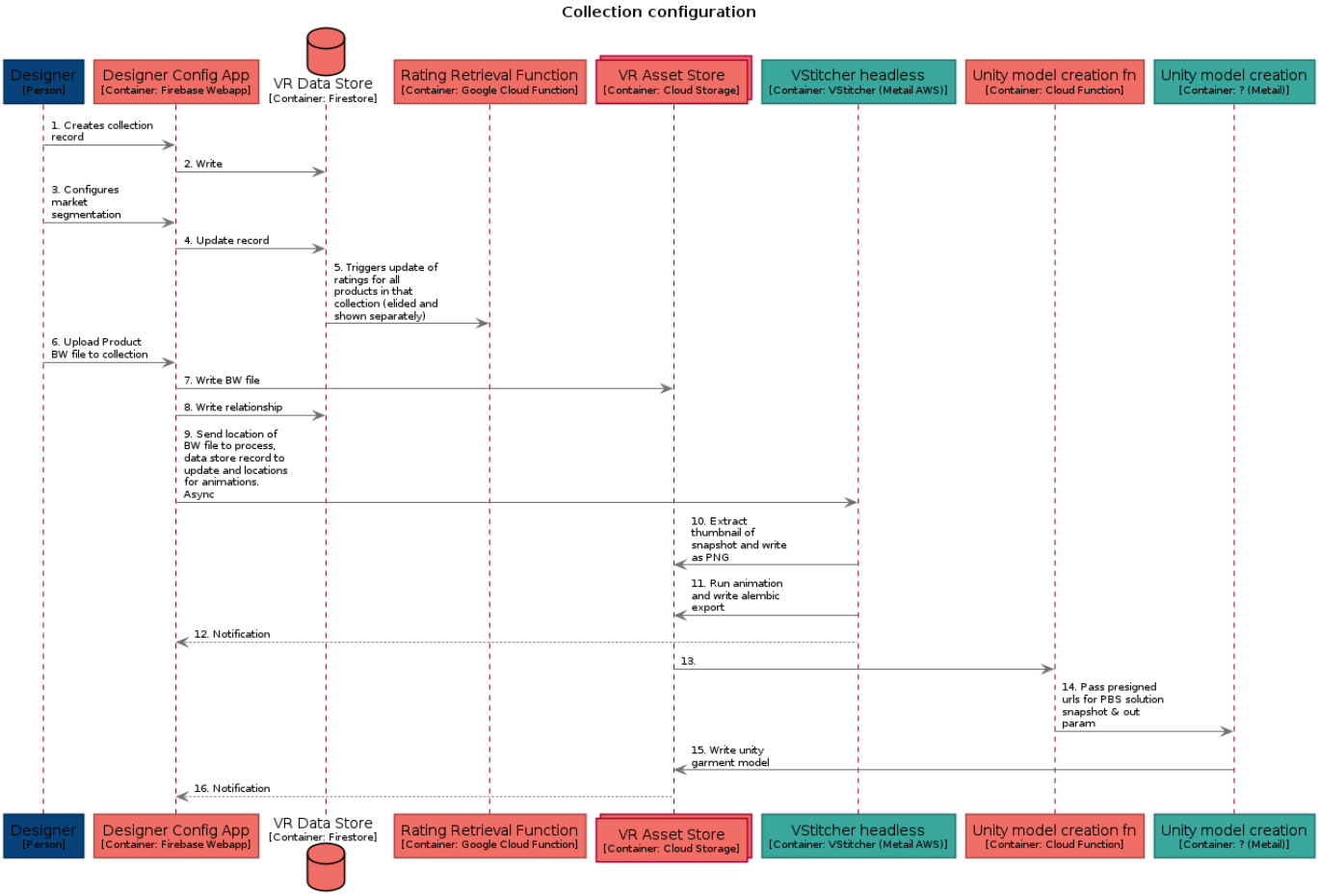
3.1.1 Fit Model Scanatar Creation



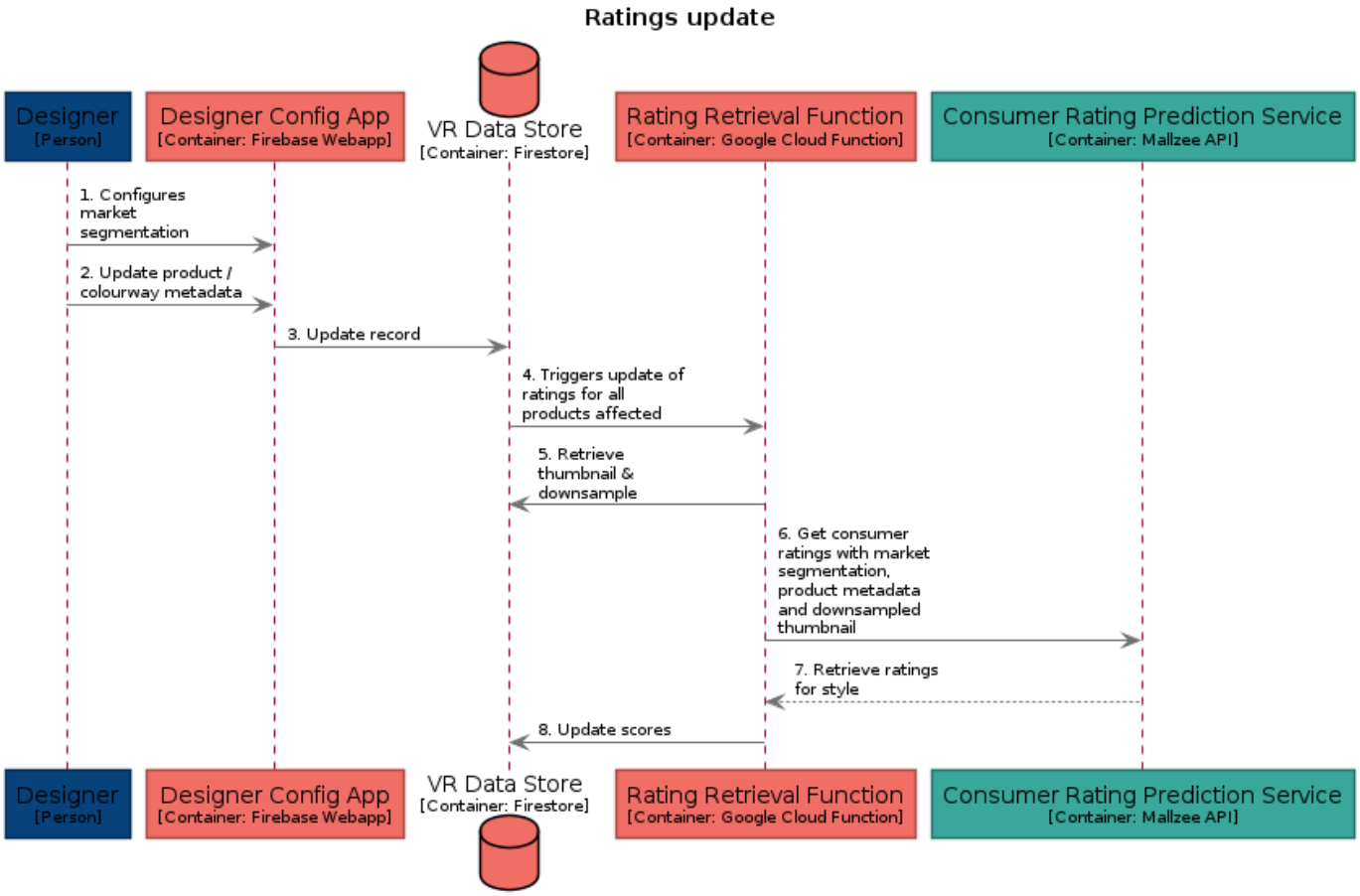
Particularly of note in this sequence diagram is the use of Cloud Storage update events to orchestrate processes where network transfer and processing times are considerable. It also uses the pattern of using pre-signed URLs for passing large file parameters to services, and as out parameters to write results to, as described in Section 4.12.

The body model creation is specifically implemented by a QuantaCorp-developed React component that acts as a client to their scanning service API, giving the user appropriate guidance and feedback to collect good images that will deliver an accurate scan result. More in Sections 4.10, 4.14 and 4.15.

3.1.2 Collection Configuration



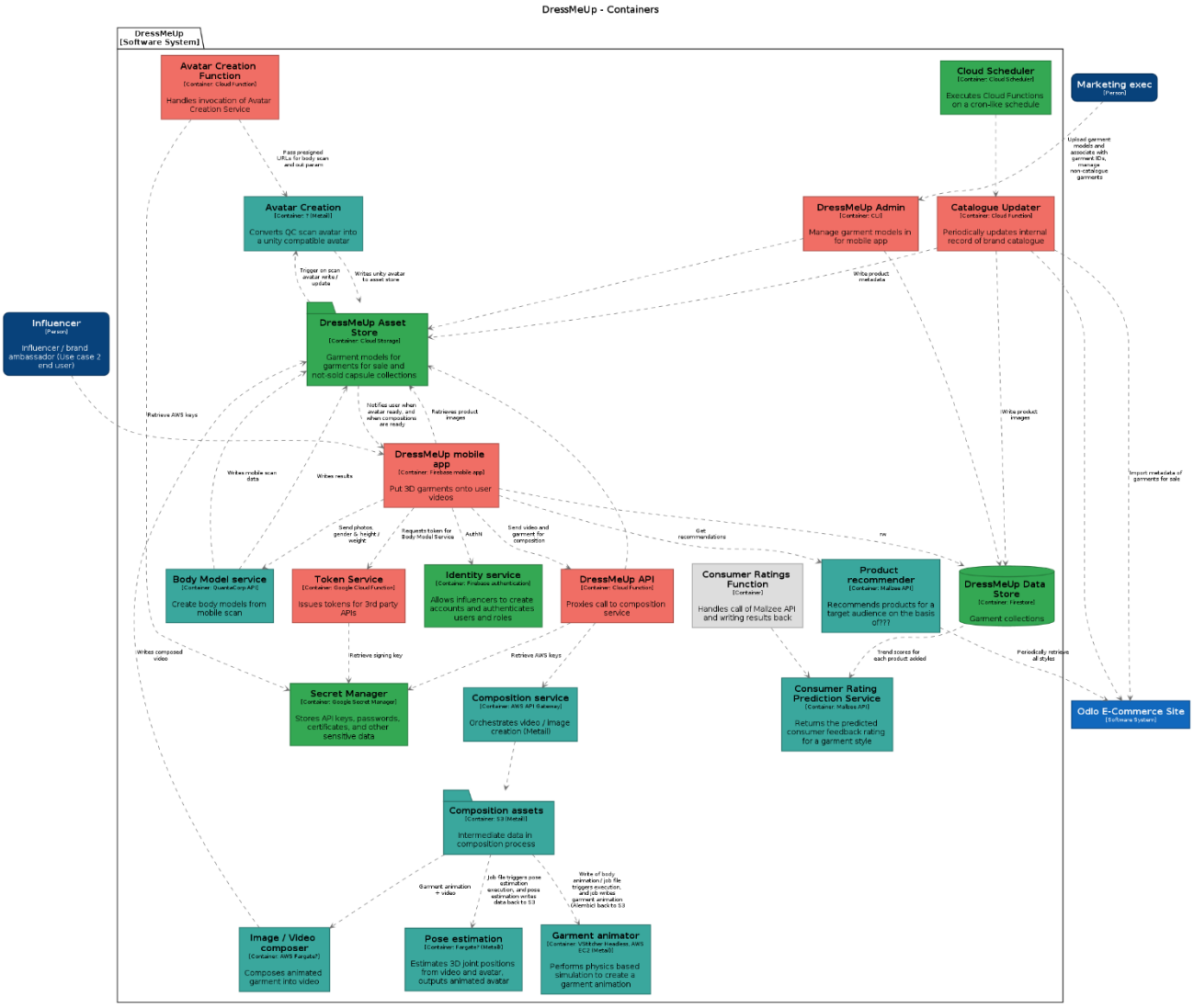
3.1.3 Ratings Update



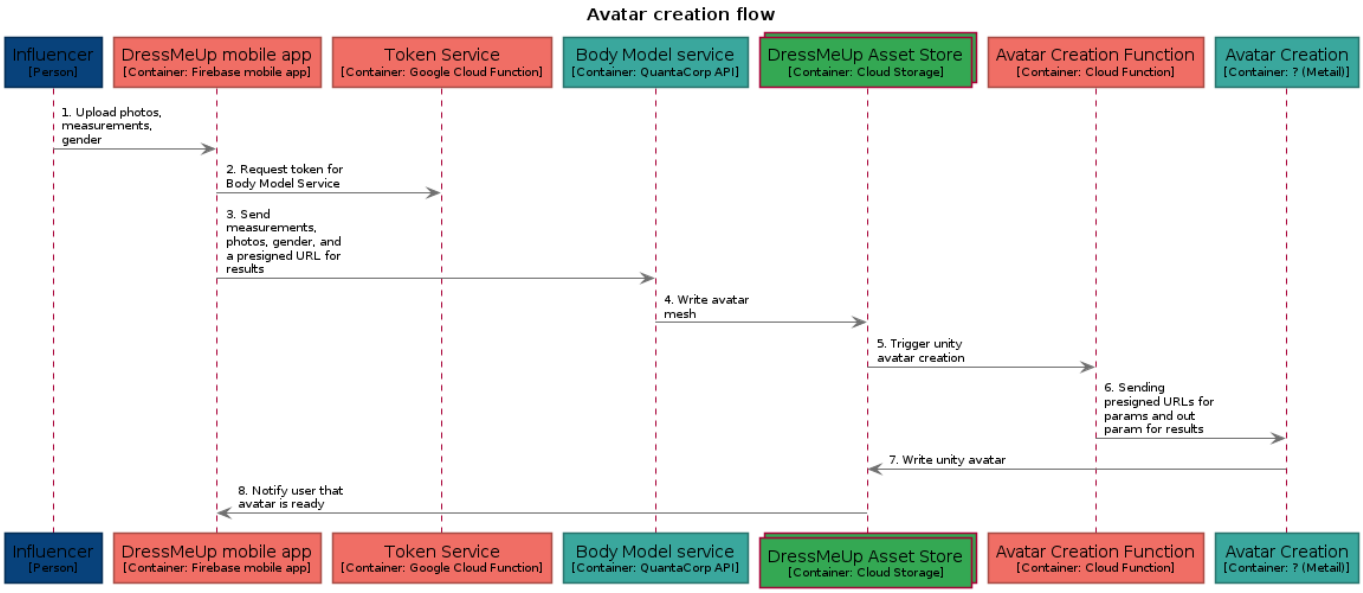
3.2 System 2 - DressMeUp System

Because of the importance of the garment animation and composition to this system, we have exposed implementation behind the Composition Service API in this diagram and subsequent sequence diagrams. Also of note is the dependency on a live product catalogue data feed from the brand's web store. Discussions with Odlo indicate that they have flexibility in the format of this feed, and our preference is for HTTPS/JSON.

Again, the system has two key user types: One role managing the data for the application, as well as the end user (an influencer) in the second role. The composition process involves a sequence of operations including pose estimation / extraction from the user's video, animation of the user's body model with the resulting animation, garment simulation on that body animation to produce a garment animation. There are a number of technology choices in how this video / image composition will be done, depending in part on the efficacy of the parts of the process. This is an open area of investigation in Work Package 2.



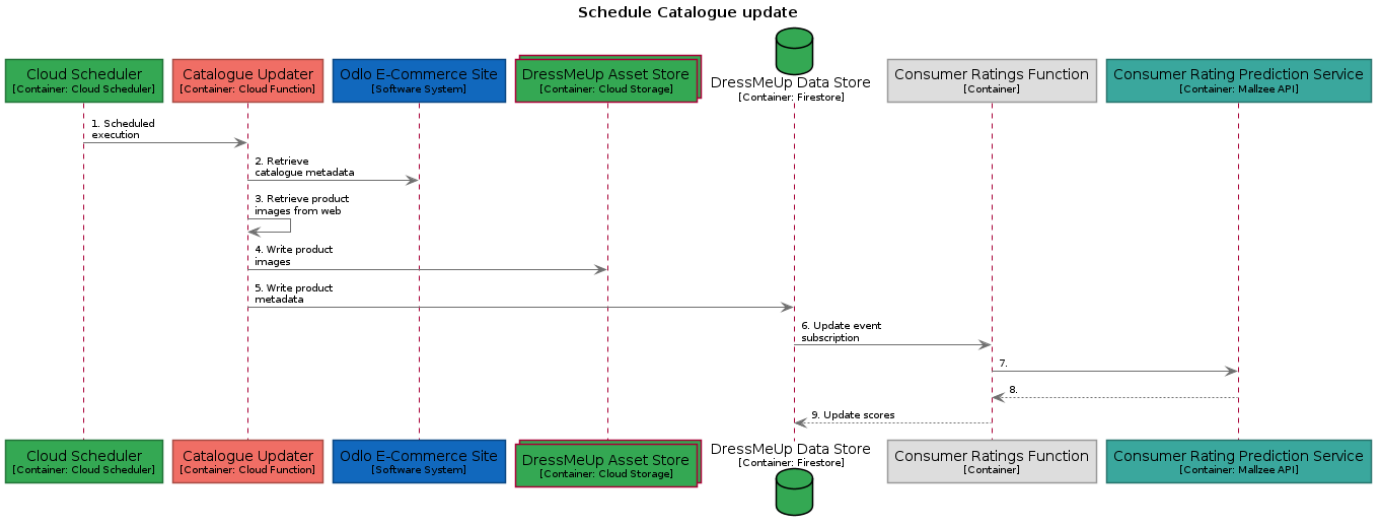
3.2.1 Body Model Creation



This use case shows another use of the QuantaCorp React SDK.

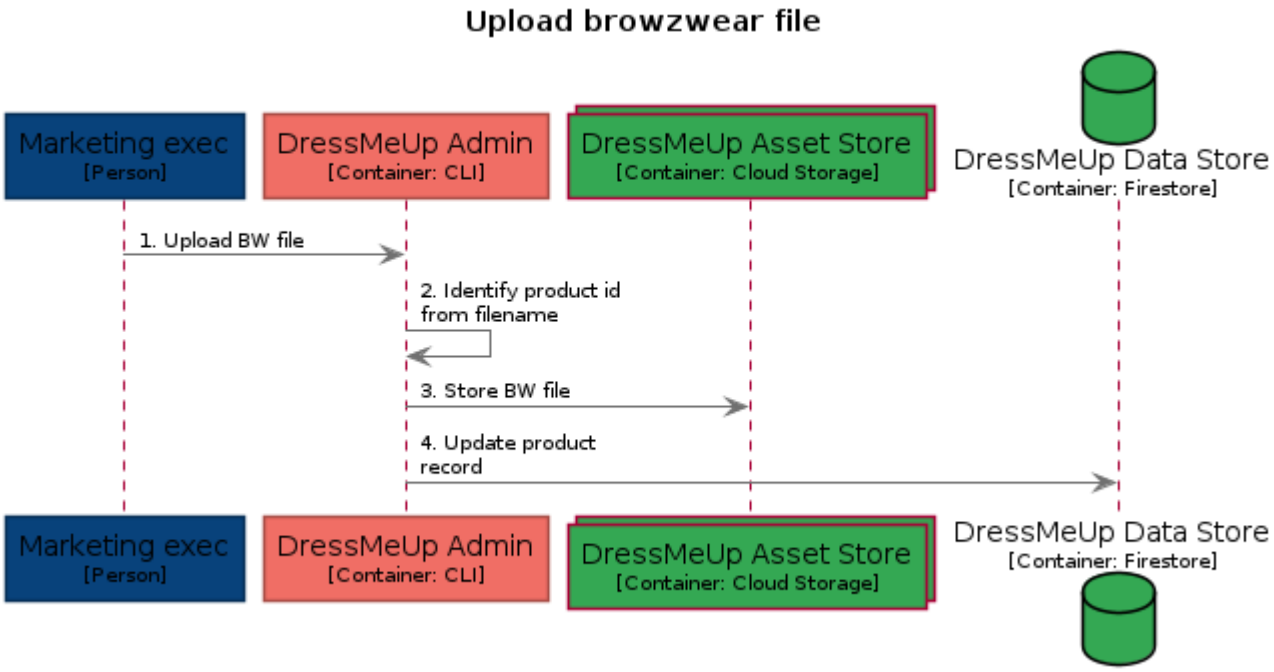
This sequence also illustrates the detail of the token mechanism used to authenticate the client app with the service layer. This will be a common pattern with most of the app-initiated sequences.

3.2.2 Catalogue Update



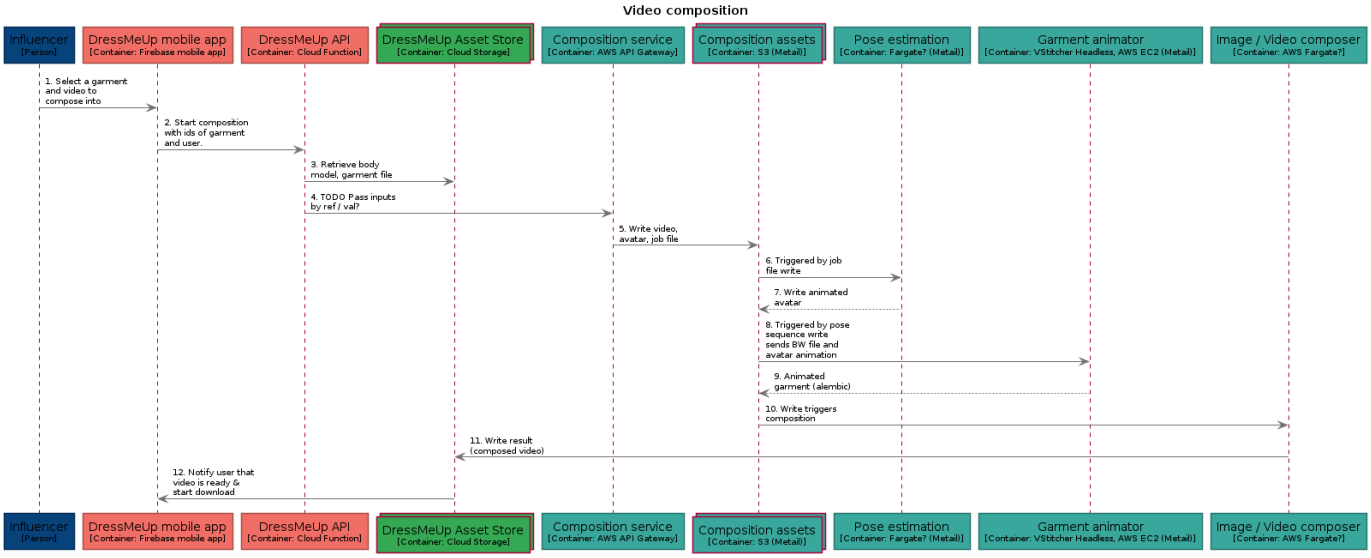
This event-driven flow leverages GCP features - the catalogue update is triggered by a cron-like scheduling service (Cloud Scheduler), and the calls to the Mallzee ratings prediction service is triggered in turn by a pub/sub event from the data store as the catalogue records are updated.

3.2.3 Browzwear File Upload

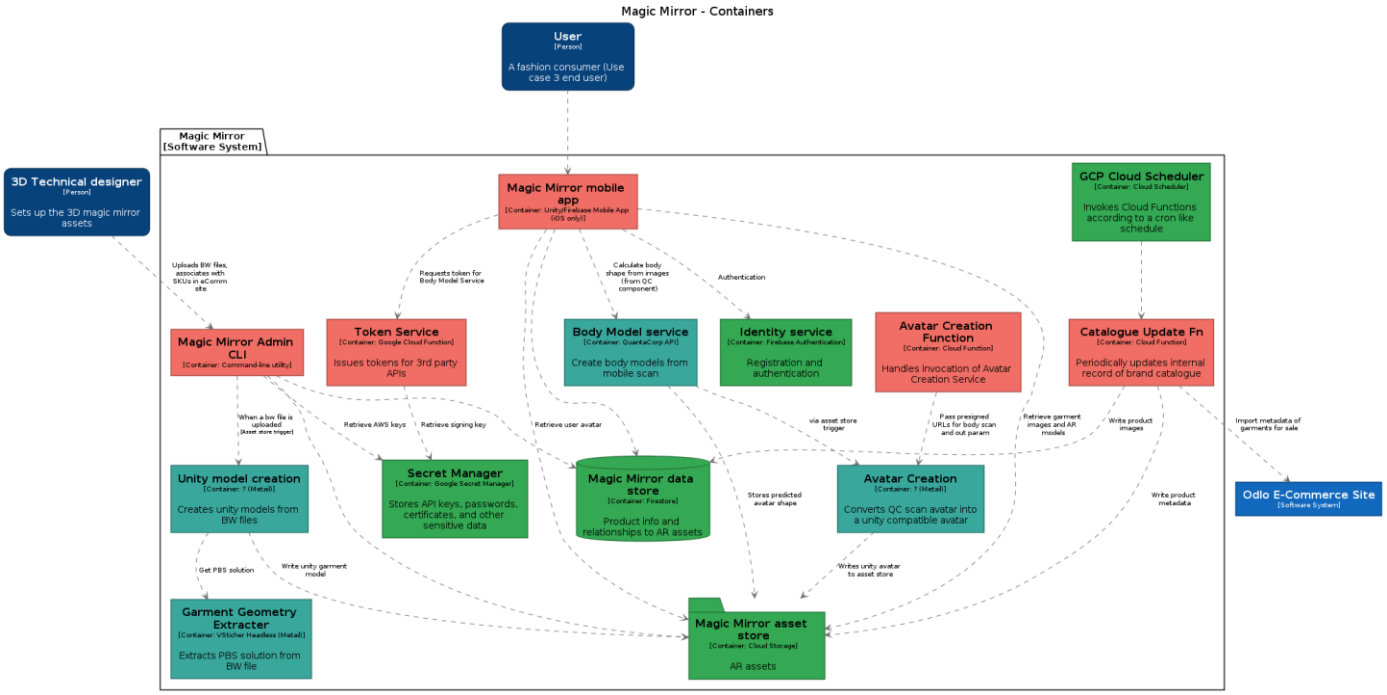


Because the current intention is to create videos by automating animations using Browzwear software directly, no pre-processing of garment data is needed. If that solution route changes, this flow could become more complex.

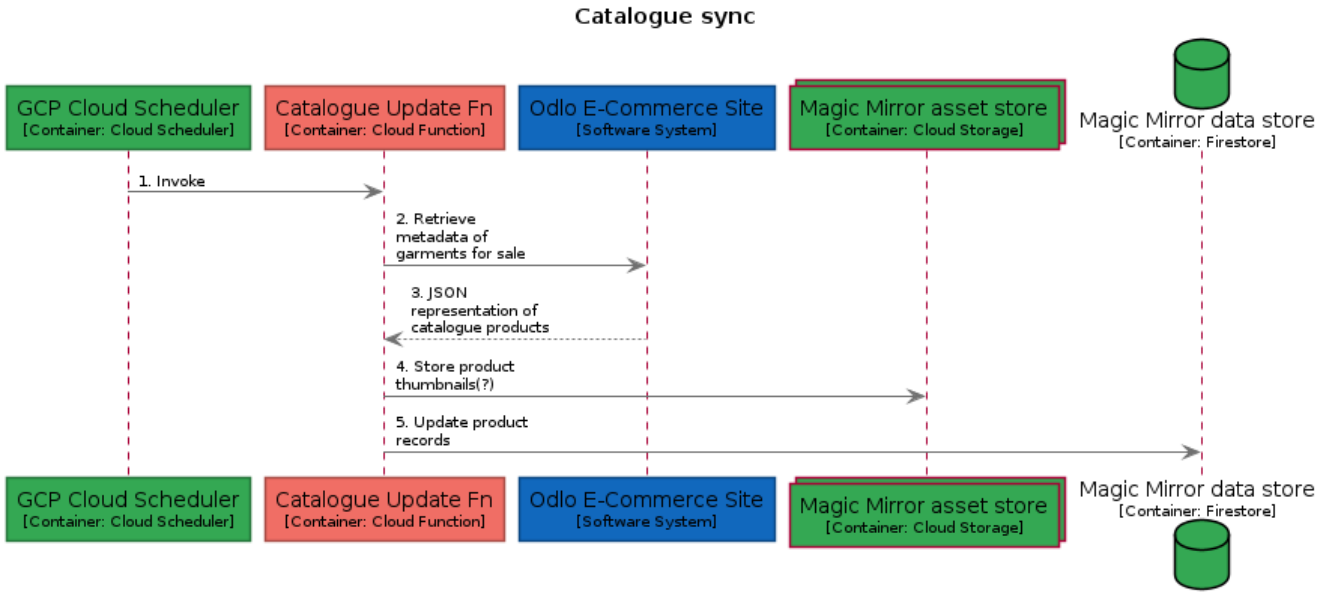
3.2.4 Video Composition



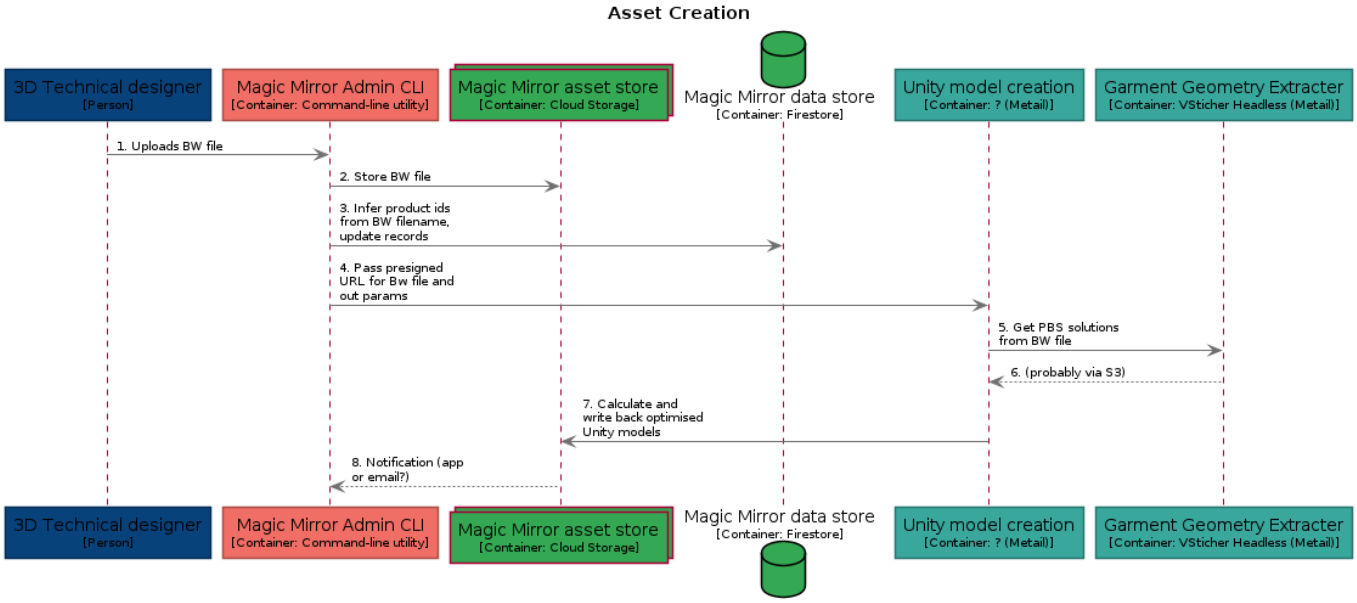
3.3 System 3 - AR try on



3.3.1 Catalogue Sync

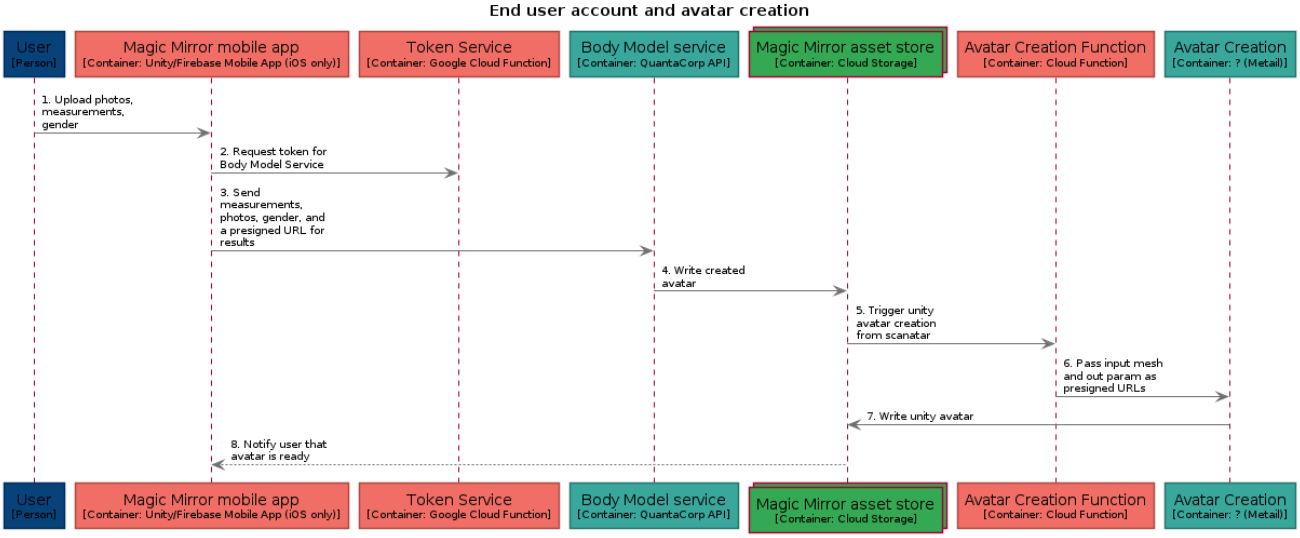


3.3.2 Asset Creation



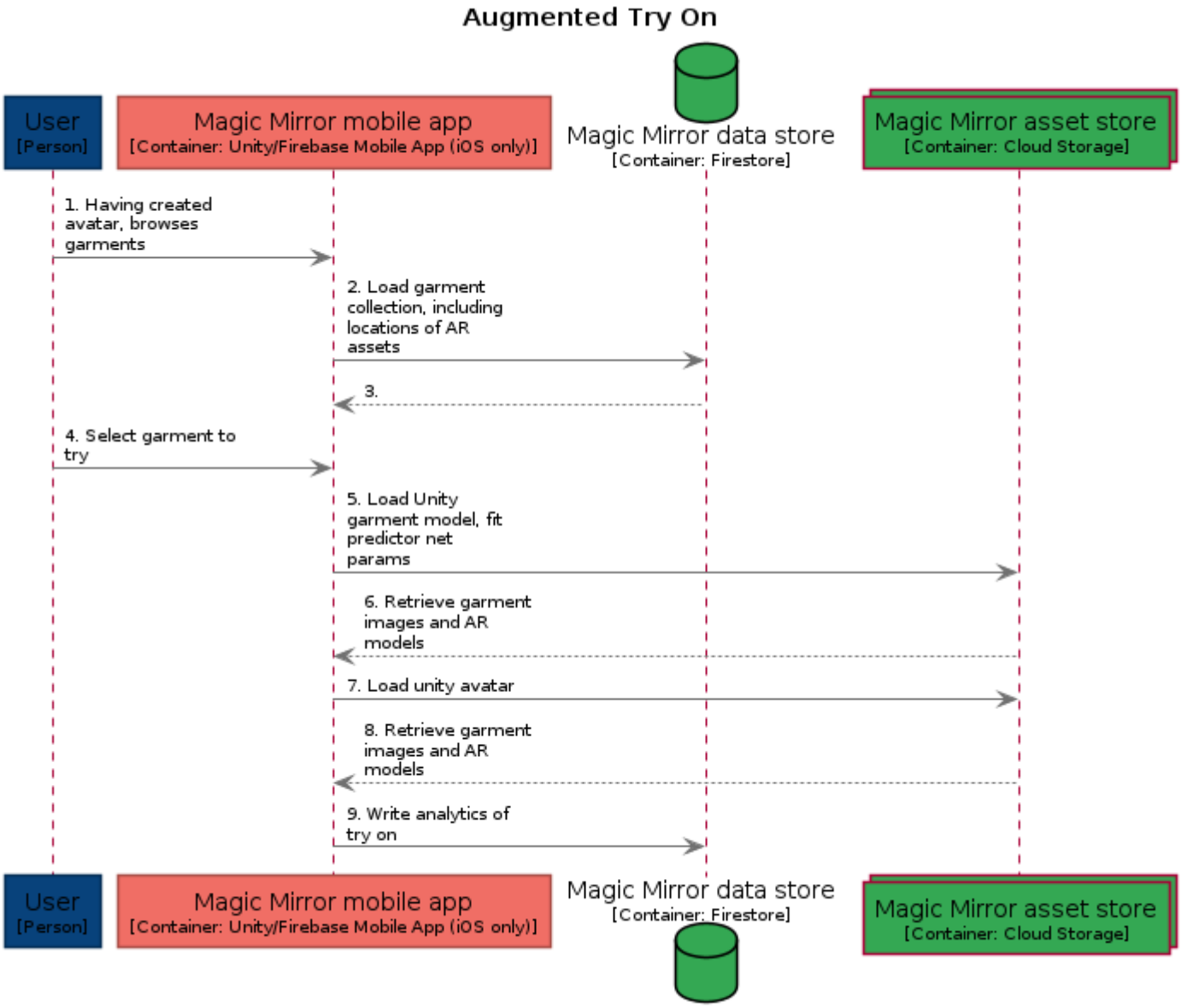
Because there is very little admin functionality needed here, and only by eTryOn project staff, we will avoid developing a thin admin Ux and instead use a Command Line Interface (CLI), as described in Section 4.16. Developing this admin functionality / enterprise integration will be part of an exploitation plan.

3.3.3 Account & Avatar Creation



In this flow, the QuantaCorp SDK is one developed specifically for iOS. It performs the same function as the React SDK - guiding the user through image collection for the mobile scan, and interacting with the QuantaCorp API.

3.3.4 Augmented Try On



4 Architecture Decisions

4.1 ADR-0001 Record Architecture Decisions

Date	2020-11-03
Status	Accepted

4.1.1 Context

We need to record the architectural decisions made on this project.

4.1.2 Decision

We will use Architecture Decision Records, as described by Michael Nygard.

4.1.3 Consequences

See Michael Nygard's article, linked above. For a lightweight ADR toolset, see Nat Pryce's adr-tools.

4.2 ADR-0002 Use C4

Date	2021-05-11
Status	Accepted

4.2.1 Context

The eTryOn project will consist of a number of user-facing applications, server-side components for data storage, processing, and retrieval, and interactions with third-party APIs and supporting systems for authentication and authorization. It will be important to document these systems and their interactions to align everyone's understanding of what we plan to build and explain how the system as a whole will work.

The software architect Simon Brown created a structured model for visualizing software architecture, the C4 model, which decomposes a system into containers and components and shows their relationships with each other and with their users. It allows the system to be viewed in a hierarchy:

- Context diagrams (level 1): they show the system in scope and its relationship with users and other systems;
- Container diagrams (level 2): they decompose a system into interrelated containers. A container represents an application or a data store;
- Component diagrams (level 3): they decompose containers into interrelated components, and relate the components to other containers or other systems;

- Code diagrams (level 4): they provide additional details about the design of the architectural elements that can be mapped to code. C4 model relies at this level on existing notations such as Unified Modelling Language (UML) or Entity Relation Diagrams (ERD).

(Reference: C4 Model).

He also provides the Structurizr tooling that allows you to express the architecture in a text-based DSL that can then be rendered at any of these levels in a number of different formats.

4.2.2 Decision

We will use the C4 Model to describe our system architecture.

We will use the Sturcturizr DSL to express our architecture diagrams.

We will use the Structurizr CLI and PlantUML to generate diagrams from the DSL.

4.2.3 Consequences

Using a text-based DSL for the diagrams allows them to be kept under version control and facilitates collaboration.

The simple DSL also allows for rapid iteration as our understanding of the requirements evolves, making it easier to keep the architecture diagrams up to date.

Command-line tooling allows us to generate images from the DSL in a script or CI/CD pipeline. These images are then easily included in Markdown pages and other documents.

4.3 ADR-0003 Use GitLab

Date	2021-05-11
Status	Accepted

4.3.1 Context

We should keep source code and documentation for the eTryOn project under version control. Most developers these days are familiar with git, a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Although git is a distributed system, most projects use a centralised hub to host their repositories and facilitate collaboration.

In addition to repository hosting, we will need a platform for building and testing any software we develop. Running tests and builds in a controlled environment (usually a Docker container) ensures reproducibility, and avoids “works on my machine” syndrome.

Two popular platforms for hosting git repositories are GitHub and GitLab; both offer free plans for small teams and provide facilities for continuous integration (automated testing and building) and collaboration (issue tracking, merge requests and review).

4.3.2 Decision

We will use GitLab to host our git repositories.

We will use GitLab CI to build and test our software.

4.3.3 Consequences

We will have a central place to host repositories where everyone on the team can find them.

We can automate building and testing of software in Docker containers (or, if necessary, custom runners).

We can use GitLab to manage and review merge requests, facilitating collaboration on shared projects.

We also have the option of using GitLab's DevOps tooling to help with deploying software and cloud infrastructure.

Although GitLab's basic plan is free, it comes with limited storage and CI/CD minutes. If we exceed these quotas we may need to purchase additional resources.

4.4 ADR-0004 Use Hugo

Date	2021-05-11
Status	Accepted

4.4.1 Context

We need to publish technical documentation for the project as it progresses (for example, architecture diagrams, use case details, and architecture decision records). We should make it as easy as possible for developers to write documentation and keep it up to date.

In ADR-0003 we describe how we will use GitLab to host source code repositories. If we choose a text-based format for our documentation, this can also be maintained in a git repository. Markdown is one such format: it is a simple and easily-readable plain text format that can be transformed to structured HTML.

Hugo is a popular static site generator that uses templates to transform a collection of Markdown files, images, and other media into a tree of static files that can be served by any web server.

4.4.2 Decision

We will use the Hugo static site generator.

We will host technical documentation on GitLab Pages.

We will configure a GitLab CI pipeline to regenerate the site and publish changes when they are pushed to the main branch.

We will configure a GitLab CI pipeline to generate a test version of the site for each merge request, to facilitate review.

4.4.3 Consequences

Maintaining documentation in a plain text format in a git repository makes it easy for developers to write and update using familiar tools.

Changes can be made to the live site by making a merge request against the main branch and merging as soon as they have been reviewed.

Collaborators can run hugo locally to test changes before pushing to the shared repository.

4.5 ADR-0005 Use APIs

Date	2021-05-11
Status	Accepted

4.5.1 Context

Some functionality for the eTryOn project will be provided by commercial companies in the consortium (QuantaCorp, Mallzee, and Metail). These companies will not in general want to make details of their implementations public, but instead present a “black box” interface.

4.5.2 Decision

We will access services from partner companies through HTTP-based APIs.

4.5.3 Consequences

We need to ensure these third-party APIs are documented so they can be consumed by eTryOn applications.

Partner companies do not need to disclose implementation details.

The systems behind these APIs can be deployed and updated independently of eTryOn project resources, so long as API-compatibility is maintained.

Control of these services remains in the hands of each partner company. Note that this implies some risk to the project as, by definition, these services are not under the control of the project.

4.6 ADR-0006 Use an Application Development Platform

Date	2021-02-09
Status	Accepted

4.6.1 Context

Mobile and web application development are increasingly achieved by using high level services provided by cloud service providers like Google Cloud Platform (GCP) or Amazon Web Services (AWS). This has led to the development of SDKs and tooling that make it easier to compose these services into working applications through simple provisioning and client-side support libraries. Essentially they provide a path of least resistance for many concerns in application development for functions from object storage to application deployment, analytics to authentication.

The two application development platforms we considered are AWS Amplify and Google Firebase. Both of these make application development easier and are backed by rich service offerings in their respective cloud platforms.

Unity game engine will be the main development platform for the eTryOn project so we created two minimal Unity applications using the aforementioned toolkits to compare the two solutions in terms of ease of use, support, and capabilities. The demo applications use simple operations, namely an authentication process and communication with cloud storage for upload and download operations.

- Firebase was easy to set up, and has an up-to-date supported SDK for Unity.
- The aws.net SDK has support for Unity but requires a tedious and error-prone set-up process.
- The Firebase platform is easier to use with a much shorter learning curve when it comes to typical backend services such as authentication, analytics, cloud storage and databases.
- The AWS SDK offers more functionality, which at the moment do not seem to be of any particular use for our case.
- There are more resources (such as documentation and community guides) available for the Firebase than for Amplify.

Note: Amazon’s Unity mobile SDK has not been mentioned so far since it has not been updated for 5 years and is now considered deprecated. The suggested Unity SDK from Amazon is the aws.net SDK considered above.

4.6.2 Decision

We will use Firebase for the eTryOn project.

4.6.3 Consequences

The documentation and community resources for Firebase will allow developers to get up and running quickly.

We will be able to use up-to-date and supported SDKs to consume cloud services in our Unity and Javascript applications.

The tooling and documentation available for Firebase will free up developers to focus on core facets of the applications.

Use of Firebase implies that we will use GCP-backed services for authentication (Firebase Auth), object storage (Cloud Storage), and NoSQL database (Cloud Firestore).

Consuming cloud services directly (via the provided SDK) means we do not need to write dedicated APIs for simple things like object and metadata storage.

4.7 ADR-0007 Isolation of Use Cases

Date	2021-05-11
Status	Accepted

4.7.1 Context

The project will build applications and deploy infrastructure for three different use cases:

- UC-1: VR Designer
- UC-2: Dress Me Up App
- UC-3: Magic Mirror App

We would like to be free to evolve these applications independently, allowing development to proceed in parallel and enabling different routes to commercialization.

Services will be deployed to Google Cloud Platform (see ADR-0006) and are charged by usage (e.g. per API call). This makes it no more expensive to give each use case its own Firebase project and deploy resources independently for each project. During development, this can be done using the Firebase CLI but for a more controlled production environment we might prefer to use something like Terraform to define infrastructure as code.

4.7.2 Decision

We will isolate the infrastructure and deployment of resources for each use case.

4.7.3 Consequences

Applications can be developed independently, in parallel.

Applications can be deployed independently of each other.

Applications may consume the same third-party APIs (see ADR-0005).

It will be easier to write authorization rules for cloud resources.

4.8 ADR-0008 Use Obi Cloth for Real-time Cloth Physics

Date	2021-02-19
Status	Accepted

4.8.1 Context

The main requirements considered for this decision are: accuracy, photorealism, and interactivity. There is always going to be a tradeoff between interactivity and the other two.

4.8.2 Requirements per use case

Overview of the different requirements depending on the use case:

- (UC1) Designer app. For VR, interactive speeds are, so a game engine model works well. However, the domain also requires accuracy. If we go with a series of fixed animations and poses, we can store the 3D animations of the accurate garment simulation from VStitcher in Alembic format (support added to VStitcher in version 2021.1). For increased photorealism, we could have the option to ray-trace fixed camera angles, and even produce EcoShots imagery for those.
- (UC2) DressMeUp app. The most important requirement is Photorealism because the final image must be worth sharing. We aren't constrained by interaction speed. The simulation and rendering can happen offline. The cloth simulation can be done in VStitcher, and the rendering can be ray-traced with V-Ray. The final render can be composed with the user's photograph using Metail's EcoShot, which should give the most photorealistic look.
- (UC3) Magic Mirror app. Interactivity is key, but we want to raise accuracy and photorealism as much as we can. Without the accuracy and photorealism requirements, we could target a platform like Snapchat's Lens Studio, that provides AR body tracking.

This ADR focuses in UC3, although the decision affects UC1 as well.

4.8.3 Platform comparisons

Game engines offer what we need in terms of interactivity. In terms of photorealism, most engines have comparable results. In terms of accuracy of the cloth simulation, results differ. Although most engines use NvCloth under the hood, the capabilities can be quite different. Here is a summary of the findings.

4.8.3.1 Lens Studio by Snapchat

Although Lens Studio does not offer any cloth simulation at the moment, it offers 3D body tracking out of the box. We can pre-simulate garments in VStitcher in some neutral pose and body shape, and then rig those garments. The skeleton in the rig needs to match the skeleton of Lens Studio. The garments will scale anisotropically based on the bone positions.

4.8.3.2 Unity Cloth

Unity is the strongest candidate for its ease-of-use, multiplatform support, and support for AR and VR. However, the in-built cloth simulation is mostly oriented to environment assets. See Getting Started with Cloth physics. For dressing up a character, we have different options:

- A fully-skinned garment, like in the Snapchat approach. See How to make clothes animate along a character.
- Have a mixture of skinned and unskinned meshes. See How to poncho as an example. The challenge of this approach is having colliders that represent the body accurately. For speed constraints, mesh colliders can't be used in cloth simulation, so we need to approximate the geometry with capsules and spheres.
- For cloth pieces, you can paint the vertices of the cloth mesh to tell the engine how much you want it to be affected by physics, where a value of 0 keeps the vertex in place.
- Use Spring Joints, which join two rigid bodies together as if they were connected by a spring. Check Spring bone vs Unith cloth.

4.8.3.3 Magica Cloth

This is a plugin for Unity that provides a faster solver and it integrates the components mentioned above in a single framework. That is, you can have a garment file that combines skinned geometry (e.g. pre-simulated shirt torso), with unskinned pieces for pure cloth simulation (e.g. a skirt), and spring bones.

However, it doesn't seem to add anything that can't already be done with Unity.

4.8.3.4 Amazon Lumberyard

Amazon Lumberyard has a few extra things useful for cloth simulation:

- Cloth simulation can be applied to skinned meshes as well, so we can have a hybrid model. Similar to Unity's weight painting, you paint the weights of the motion constraints to define how far away a cloth vertex can move. In this case, how far from its position *after* skinning, so the vertex is also affected by skinning. The weights are normalised between 0 and 1, and then there's a maximum distance value per cloth piece, defined in metres.
- The weights are stored as color channels in the FBX file, which it's something we may want to consider if we want to export a single file.
- There is also a backstop value which is a cheap substitute for colliders. Instead of having colliders, we define how much the vertex can move in the normal direction of the mesh (a negative value for the vertex of a sleeve would allow it to penetrate the arm).
- Skinning in Lumberyard uses dual quaternion by default, which it's useful to avoid bulging and candy-wrapper artefacts in skinned meshes. It is supported in mobile as well. Read Introduction to skinning and 3D animation.
- It automatically automatically stitches and creates a proxy geometry for simulation on garment import. The Level-of-Detail of the proxy geometry can be set as well. For

rendering, the proxy geometry is replaced by the original geometry, so you can have multiple unstitched materials and a higher polygon count for photorealism.

Lumberyard does not support spring joints, but those are probably irrelevant. Spring joints are a cheap way to make bouncy skirts when cloth simulation is not avaible, or it's unstable.

The main drawbacks for using Lumberyard are the lack of official AR support, the difficulty of building for multiple platforms (all major mobile and desktop platforms are supported, but the setup is not as straightforward as Unity), and build times.

4.8.3.5 Obi Cloth

Obi Cloth is a Unity plugin that adds support for the things mentioned above in the Lumberyard section, mainly support for skinned meshes, and cloth proxies. The motion constraints described earlier are called skin constraints.

The only disadvantage with respect to Lumberyard is that the skinning depends on Unity skinning, and Unity does not support dual-quaternions skinning.

However, we gain all the advantages of using Unity: fast development cycles, and good support for AR and VR.

4.8.4 Decision

We will use Unity for UC1 and UC3. For UC3 we will use Obi Cloth, which enables Unity with all the advantages seen in other engines like Amazon Lumberyard.

For UC1, we can use the same engine, but we also have the option of using Alembic 3D animation playback for increased accuracy, at the cost of some interactivity (we can still move freely inside the VR world and pause the animations at any point to inspect the garment).

We will also build a teaser app of UC3 using Lens Studio. This will use the same garment creation pipeline, but with a special skeleton for Lens Studio. It won't require us to export any physics properties, so it can be created earlier.

4.8.5 Consequences

Garment data needs to be prepared for Obi Cloth. In particular, we need to check the following:

- Do we need to pre-stitch garments beforehand, or can Obi do all the stitching?
- How do we provide the skin constraints weights?
- How do we convert other physics properties from VStitcher to Obi?

4.9 ADR-0009 Platform for Each Application

Date	2021-02-24

Status	Accepted
Superseded by	ADR-0017 Platform for Each Application revision

4.9.1 Context

The applications that will be developed are:

1. VR Designer: Desktop
2. Dress Me Up: Mobile
3. Magic Mirror: Mobile
4. VR Designer Backoffice: Web

4.9.2 Decision

The following platforms will be used to develop the applications:

1. VR Designer: Unity
2. Dress Me Up: Unity
3. Magic Mirror: Unity
4. VR Designer Backoffice: JavaScript (React.JS)

4.9.3 Consequences

Unity has support for building mobile UIs and can export applications for all major mobile platforms.

Using Unity to develop both the desktop and cross-platform mobile applications means there is less effort in learning additional platforms.

4.10 ADR-0010 QuantaCorp SDK

Date	2021-03-09
Status	Accepted

4.10.1 Context

The eTryOn project will be developing multiple applications that require an avatar to work or to enrich the user experience. The creation of this avatar starts with the capture of two images. Those two images are sent as input to the QuantaCorp pipeline. After processing, the output of that pipeline is a 3D model.

QuantaCorp’s current technology portfolio consists of an API, a web portal for B2B customers, and a mobile app for B2B customers on iOS and iPadOS.

Given that eTryOn is customer focused, we need a solution that will meet customer expectations while keeping the impact on QuantaCorp’s architecture to a minimum.

To reach a broad public, we need to expand to other platforms like the web and Android. We should avoid having to make the user open a separate app to capture the required photos. Having to switch apps can have a negative effect on the overall user experience. Because there is no direct communication channel as there is in a B2B environment, guidance during a scan becomes very important. The user will also expect the scan interface to work in the same fashion across platforms. If this experience cannot be guaranteed to be similar, guidance during a scan becomes a must.

Developers should be provided with libraries that facilitate the consumption of the QuantaCorp API, and allow them to integrate QuantaCorp's pipeline in a loosely coupled way.

4.10.2 Decision

We will create an SDK for Android, iOS and web that will consume the QuantaCorp API and will include a scan component to facilitate photo capture.

4.10.3 Consequences

By creating an SDK for the various eTryOn apps, scanning functionality will be implemented in the apps themselves. This way we avoid having to force the user to use a separate app.

The SDK will be implemented in Unity (cf ADR-0006) which could pose a small development challenge.

We reduce the complexity of the project's architecture by avoiding the integration of yet another application, and yet another system.

4.11 ADR-0011 Mallzee Api

Date	2021-05-11
Status	Proposed

4.11.1 Context

As part of the eTryOn project it requires that clothing products should be evaluated for product market fit against target demographics and provide recommendations of other products based on a given product.

Mallzee has a datapool of consumer options again 4 million+ products and has experience in attempting to predict future product performance by using consumer data captured via the Mallzee shopping app.

Mallzee is looking to provide access to these insights so that other companies can benifit from the data and models produced. This in turn allows companies to make smarter decisions about the types of products they want to produce by using the vast consumer data available via the Mallzee apps.

4.11.2 Decision

Mallzee will produce an API that will be available to the eTryOn consortium. The API will consisit of a minimum of two endpoints to provide access to the product predictions and the product recommender.

4.11.2.1 Authentication

This Mallzee API is only required to be accessed via internal services. Not directly from the client applications. We will use a simple header based authentication method using the key `x-api-key` which will contain a string access key assigned to every consumer of the API. This allows Mallzee to track usage and identify which account is accessing the API.

The eTryOn services will store the given key in the Google Secret Manager so that access can be given to any service within the platform.

4.11.2.2 Product predictions

This endpoint will require that the user send product data and target market data so that the system can predict the popularty of the product and return a confidence score of how likely that product is going to prove popular with the target demographic.

4.11.2.3 Product recommendations

This endpoint will require that the consumer of the API sends the required product information along with the end users market preferences. This system will return an array of recommended product IDs based on the information given that will allow the consumer to fetch product information on the recommended products.

These API will be detailed in OpenAPI Spec 3.0 and can be found here (Link to be provided)

4.11.3 Consequences

By creating a general purpose API we can supply product performance predictions to all applications that require it as part of the overall project.

4.12 ADR-0012 Metail Scanatar Creation

Date	2021-05-17
Status	Proposed

4.12.1 Context

The scanatars created by the QuantaCorp Body Model Service are post-processed by a Metail pipeline that performs clean-up of the scan, landmark detection, mesh fitting, and skeleton fitting. The output from the Metail pipeline is a Unity-compatible avatar that can be used in eTryOn applications. We would like the Metail pipeline to run every time a new QuantaCorp scanatar is uploaded to Cloud Storage.

4.12.2 Decision

Metail will implement an AWS Step Function to run the Metail pipeline.

This step function will:

- Read additional parameters (such as gender and height) from its invocation parameters;
- Read the QuantaCorp scan from Cloud Storage using a pre-signed URL;
- Run the Metail Scanatar Pipeline on this input;
- Write the finished scanatar to Cloud Storage using a pre-signed URL.

We will create an eTryOn service user in the Metail AWS IAM account and grant this user permissions to invoke the step function.

We will generate access keys for the AWS service user and store them in Google Secret Manager so they are available to services running in the eTryOn Google Cloud project.

We will implement a Google Cloud Function and configure a trigger to invoke this function whenever a QuantaCorp scanatar is written to Cloud Storage. This cloud function will:

- Create a record in Cloud Firestore to track the pending avatar creation;
- Create pre-signed Cloud Storage URLs for the input (QuantaCorp scanatar) and output (Unity-compatible avatar);
- Read the configured AWS keys from Secret Manager;
- Invoke the Metail Step Function and pass the pre-signed input/output URLs and additional metadata (such as the user's height and gender) as parameters.

We will implement another Google Cloud Function to be triggered when the finished avatar is written to Cloud Storage. This function will update the Cloud Firestore record to update the status of the new avatar.

4.12.3 Consequences

Providing Metail AWS credentials to eTryOn middleware functions enables them to call AWS APIs directly.

Using pre-signed URLs for input and output avoids us having to share credentials in the opposite direction.

Tracking scanatar creation status in Cloud Firestore allows client applications to show jobs in progress and be notified when the status changes.

We will have to ensure the pre-signed URLs have a long enough expiry for the Metail pipeline to complete and write its output back to Cloud Storage.

We will need a mechanism to signal an error if the Metail Pipeline cannot process the input QuantaCorp scanatar.

4.13 ADR-0013 Metail VStitcher Headless Service

Date	2021-05-17
Status	Proposed

4.13.1 Context

Several eTryOn use cases require background processing of Browzwear files to provide functionality (for example, to generate the Alembic animation file for use case 1). Processing of a Browzwear file will always be triggered by a middleware function when a Browzwear file is uploaded to Cloud Storage.

4.13.2 Decision

We will follow the pattern described in ADR 12 with a Google Cloud Function in the eTryOn account invoking a Step Function in the Metail AWS account to process Browzwear files.

Metail will implement an AWS Step Function to read input from, and write output to, Cloud Storage using pre-signed URLs. The step function will invoke the desired script to run in VStitcher Headless running on an EC2 Windows instance.

4.13.3 Consequences

We will have to define the processing required by each use case and encapsulate this as scripts that can be run by VStitcher Headless.

We will need a way to identify which script to invoke for each use case (e.g. through a naming convention in Cloud Storage).

If we are unable to secure a license to run VStitcher Headless, the step function will have to initiate a manual process where jobs are completed by a human using VStitcher Desktop.

4.14 ADR-0014 Scanatar Creation

Date	2021-05-11
Status	Accepted
Amended By	ADR-0015 Token Service for QuantaCorp API

4.14.1 Context

In ADR-0010 QuantaCorp proposes to write a cross-platform SDK (for Android, iOS and web) that will facilitate photo capture and interact with the QuantaCorp API. This document goes into more detail about the architecture supporting this SDK and the scanatar creation process.

We assume that all of the applications using the SDK will require the user to login using Firebase authentication. This gives the applications access to Cloud Storage and an ID token that can be used to interact with other APIs (including the QuantaCorp API).

We also assume that the QuantaCorp SDK will have access to client-side functionality of the Firebase SDK, as all of the client applications will use the Firebase SDK.

4.14.2 Decision

The QuantaCorp SDK will use the ID token obtained via Firebase Auth as a bearer token to authenticate with the QuantaCorp API. It is the responsibility of the QuantaCorp API to validate this token.

The application implementing the SDK will generate pre-signed URLs to store the output from the QuantaCorp system in Cloud Storage owned by eTryOn. We will use a naming scheme that stores all of the user's files under a prefix of their Firebase user ID, which simplifies the authorization rules controlling access to Cloud Storage.

The application will prompt the user to enter their height and gender and save these as metadata to Cloud Firestore.

The SDK will upload the photos required for avatar generation directly to the QuantaCorp API along with the metadata, the pre-signed URLs and the ID token.

On successful upload of metadata and photos, the QuantaCorp system will generate a thumbnail from the front view photo and a OBJ file for the scanatar.

The QuantaCorp system will use the pre-signed URLs obtained from the client to write the output thumbnail and OBJ file directly to the user's eTryOn Cloud Storage area.

Successful creation of the OBJ file in Cloud Storage will trigger a Cloud Function to hand off the OBJ file to Metail's systems for rigging and creation of the final scanatar. This Cloud Function will read the user's height and gender from Firestore to pass to the Metail API.

The Metail system will also use a pre-signed URL to fetch the input from the user's Cloud Storage and write back the rigged scanatar.

4.14.3 Consequences

- The eTryOn systems will not handle the user's photos, so we avoid unnecessary handling of personally identifying data.
- Once the thumbnail and scanatar have been written successfully to the user's eTryOn Cloud Storage, the input photographs can be deleted from QuantaCorp's systems.
- Terms and conditions for applications using the QuantaCorp SDK to generate an avatar must make explicit the retention period for the user's photos, whether they are deleted immediately on completion of avatar creation or retained; if retained, the terms and conditions must state the reason for retention.
- Using pre-signed URLs minimizes the trust between systems and removes the need to issue service credentials.

- The Cloud Function invoking the Metail API may need credentials as this is the one place we don't have access to the user's ID token.

4.15 ADR-0015 Token Service for QuantaCorp API

Date	2021-05-14
Status	Proposed
Amends	ADR-0014 Scanatar Creation

4.15.1 Context

The QuantaCorp API needs to be able to validate that calls are coming from a legitimate eTryOn user. In ADR 0014 we proposed using the ID token returned by Firebase Auth as a bearer token to authenticate to the QuantaCorp API, but as this is the same token used to authenticate to Firebase services it should not be exposed to a third party API.

4.15.2 Decision

We will implement a Google Cloud Function to generate short-lived tokens with scope limited to the QuantaCorp API.

These will be JSON Web Tokens containing the ID of the authenticated user in the `id` field and `QuantaCorp` in the `audience` field, and will be valid for 15 minutes.

We will generate a key pair for signing and validating the tokens. We will share the public key with QuantaCorp so they can verify the signature.

4.15.3 Consequences

Applications using the QuantaCorp SDK will have to retrieve a token (by calling the Cloud Function) before interacting with the QuantaCorp API. They will authenticate to the Cloud Function using the ID token retrieved from Firebase Auth.

The QuantaCorp API will be able to identify the user from the `id` field in the token claims. This claim can be validated by verifying the token signature.

4.16 ADR-0016 Authentication and Authorization

Date	2021-05-14
Status	Proposed

4.16.1 Context

In ADR 0006 a decision was made to use the Firebase platform to build the client-facing applications. This platform includes the Firebase Auth component which provides backend services, SDKs, and ready-made UI libraries for account management and authentication.

Even using a drop-in framework like this we need to understand how accounts will be created and how permissions will be granted.

We identified the following requirements for the three use cases:

4.16.1.1 UC1: VR Designer

- Account creation and permission granting will be done by eTryOn staff.
- Data configuration will be done by Brand staff.
- No creation of accounts possible through Ux.

4.16.1.2 UC2: Dress Me Up

- No brand staff accounts needed.
- eTryOn staff accounts and permissions will be created manually.
- Data configuration will be done through CLI by eTryOn staff.
- End users (influencers) need a registration Ux for account creation.
- No permissions granting / account approval step needed for end users.

4.16.1.3 UC3: Magic Mirror

- No brand staff accounts needed.
- eTryOn staff accounts and permissions created manually.
- Data configuration done through CLI by eTryOn staff.
- End users (shoppers) need a registration Ux for account creation.
- No permissions granting / account approval step needed for end users.

4.16.2 Decision

We will implement UI for end users (influencers) to sign up for an account in the Dress Me Up app.

We will implement UI for end users (shoppers) to sign up for an account in the Magic Mirror app.

We will create IAM accounts with appropriate permissions for eTryOn staff to perform administrative tasks (data configuration for Dress Me Up and Magic Mirror apps, brand user account creation for VR Designer app).

The applications will be single tenant. We will not implement group membership and management within an app.

4.16.3 Consequences

Using Firebase Auth and the supported SDKs makes it easy to implement UI components for sign-up and sign-in, and for the client applications to interact with server-side Firebase components.

We will have to define authorization rules for Cloud Storage and Cloud Firestore to control user access to resources.

When Cloud Functions are called directly from one of our apps, the auth token will be validated automatically but we may need to implement authorization checks in the function itself.

It is not clear from the Firebase Auth documentation that we will be able to prevent a malicious user from accessing Firebase backend services to self-register an account, even when we do not provide a UI for this. If it is not possible to disable those backend APIs we will have to ensure that accounts created in this way have no privileges within the system. This is an area requiring more research.

4.17 ADR-0017 Platform for Each Application revision

Date	2021-05-18
Status	Proposed
Supersedes	ADR-0009 Platform for Each Application

4.17.1 Context

In ADR 0009 a decision was made to use specific platforms for each application.

There has been a change in the usage of the second application “Dress Me Up app”. Instead of a mobile app it is now a web app. So for the implementation we will be using JavaScript (React.JS).

4.17.2 Decision

Subsequently the platforms that will be used to develop the applications are now the following:

1. VR Designer: Unity
2. Dress Me Up: JavaScript (React.JS)
3. Magic Mirror: Unity
4. VR Designer Backoffice: JavaScript (React.JS)

4.17.3 Consequences

We have to take into account permissions to use a laptops camera or a webcam, when a user is using the application through a web browser instead of a mobile device.