**MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

GARMENTO - A modular Virtual Clothes Try-on System for Fashion Manufacturers

by

Do Hai Binh

A thesis submitted in conformity with the requirements  
for the degree of Master of Software Engineering

© Copyright by Do Hai Binh 2024

**MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

GARMENTO - A modular Virtual Clothes Try-on System for Fashion Manufacturers

by

Do Hai Binh

A thesis submitted in conformity with the requirements  
for the degree of Master of Software Engineering

Supervisor:

1. Assoc. Prof. Phan Duy Hung
2. Dr. Vu Thu Diep

© Copyright by Do Hai Binh 2024

GARMENTO - A modular Virtual Clothes Try-on System for Fashion Manufacturers

Do Hai Binh

Degree Master of Software Engineering

FPT University

2024

Abstract

In response to the rapidly changing demands of consumers and industry stakeholders in the fashion technology sector, the development of innovative solutions is essential. This work presents GARMENTO, a customizable B2B virtual clothes try-on system tailored to meet the evolving requirements of the fashion industry. The study outlines the investigation of business needs related to virtual try-ons, evaluates web service architecture styles to guide GARMENTO's design, identifies key functionalities for a minimum viable application, and explores implications and future enhancements. By integrating insights from business requirement analysis and architectural assessments, GARMENTO aspires to be a benchmark for fashion brands aiming to adopt virtual try-on systems, thereby advancing industry standards and fostering innovation. Continuous improvements and future explorations will ensure GARMENTO remains a leader in virtual try-on technology, enhancing user experiences and shaping the future of fashion retail.

Acknowledgments

I would like to express my gratitude to my supervisors, Assoc. Prof. Phan Duy Hung and Dr. Vu Thu Diep, for guiding me on initial topic formulation and research direction. This work would not have been possible without their unwavering support. I would also like to acknowledge the help of my colleagues, especially my technical advisor, Mr. Do Anh Tuan, who gave me assistance on technical aspects for an enterprise application and also brought smiles to me during the hard times when I carry out this research. Finally, I would like to dedicate this work to my family for the endless love and support that they provided to me.

Table of Contents

[Acknowledgments 2](#_Toc165800854)

[Table of Contents 3](#_Toc165800855)

[List of Figures 4](#_Toc165800856)

[Chapter 1 Introduction 5](#_Toc165800857)

[1 Problem & Motivation 5](#_Toc165800858)

[2 Existing work 6](#_Toc165800859)

[3 Problem & Motivation 7](#_Toc165800860)

[Chapter 2 Background Study 9](#_Toc165800861)

[1 Business of Fashion Brands 9](#_Toc165800862)

[2 Modular Software Architecture 10](#_Toc165800863)

[3 ML Models in Production 11](#_Toc165800864)

[Chapter 3 System Design 13](#_Toc165800865)

[1 Business Flow 13](#_Toc165800866)

[2 Design Diagrams 14](#_Toc165800867)

[2.1 Context Diagram 15](#_Toc165800868)

[2.2 Container Diagram 16](#_Toc165800869)

[2.3 Component Diagram 18](#_Toc165800870)

[Chapter 4 Implementation & Discussion 21](#_Toc165800871)

[1 Try-On & Catalog Module 21](#_Toc165800872)

[2 Try-on Integration into an E-commerce Application 26](#_Toc165800873)

[3 MLOps Implementation 29](#_Toc165800874)

[4 Microservice Implementation 31](#_Toc165800875)

[5 Discussion, Future work & Conclusion 34](#_Toc165800876)

[References 38](#_Toc165800877)

List of Figures

[Figure 1. The design process in the Textile & Clothing industry. 10](#_Toc165824739)

[Figure 2. GARMENTO system main flow 14](#_Toc165824740)

[Figure 3. Context Diagram of the GARMENTO Virtual Try-on system. 15](#_Toc165824741)

[Figure 4. The Container Diagram of the GARMENTO Virtual Try-on System. 17](#_Toc165824742)

[Figure 5. The Component Diagram of GARMENTO Virtual Try-on System. 19](#_Toc165824743)

[Figure 6. User Interface of Try-On function. 21](#_Toc165824744)

[Figure 7. Google sign-on screen. 22](#_Toc165824745)

[Figure 8. Obtaining service-level credentials for the Try-On & Catalog module. 23](#_Toc165824746)

[Figure 9. Performing virtual try-on with preset. 24](#_Toc165824747)

[Figure 10. Catalogue status transitions. 25](#_Toc165824748)

[Figure 11. Virtual try-on plugin integrated into an example e-commerce web application. 27](#_Toc165824749)

[Figure 12. Sequence diagram for E-commerce Application integration. 28](#_Toc165824750)

[Figure 13. Model Registry’s Sequence Diagram. 29](#_Toc165824751)

[Figure 14. Service decomposition in the GARMENTO system 31](#_Toc165824752)

[Figure 15. High-level deployment diagram (non-UML) of the GARMENTO prototype. 33](#_Toc165824753)

1. Introduction

# Problem & Motivation

Machine Learning and its descendant, Deep Learning, have been out for a while and are becoming increasingly important not only for general purposes but also in many businesses, which is proved by the expansion in popularity of ChatGPT, one of the most recent AI-powered applications. AI products are developed to aid humans in the large-scale processing of common types of daily data, such as language and images. Deep learning models have empowered image processing products to not only recognize patterns but also synthesize natural-looking images from other images or even from text. Thanks to the advances in generative image processing deep learning models, many once impossible tasks are now made feasible. One example is the Virtual Try-on task, which includes not only clothes or fashion items but also make-up patterns. There has been a lot of research on multiple types of Virtual Try-on, ranging from glass try-on, and clothes try-on to make-up try-on. Much of this research has been transformed into real-world applications, for example, Glass Try-on by Eyeconic [1] in an e-commerce website, or make-up try-on mini-apps by Maybelline [2] or L’Oreal Paris [3]. These applications demonstrated how virtual try-ons help brands’ customers see whether the products they are going to buy online will fit them perfectly, therefore improving customers’ confidence in e-shopping. In addition to customer-side applications, Virtual Try-on may also apply to support fashion businesses in designing their products. However, applications in this regard are scarce, with only a few notable names, such as Reactive Reality company [4]. Reactive Reality’s product is a generally available software package companies can subscribe to and use out of the box without much customizability. This work will delve into designing a modular, customizable fashion try-on system, GARMENTO, which fashion brands can integrate into their existing ecosystem, taking fine-grained control over infrastructural and business concerns. In the design, we will compare microservice and modular monolithic architectures to determine which style is more suitable for the system. Additionally, the design will take into account the role of MLOps (Machine Learning Operation) in continuously improving and integrating the try-on deep learning model.

# Existing work

Implementing a fashion Virtual Try-on system requires knowledge of how machine learning models work and which Virtual Try-on models are currently high performers. According to a survey by Sun et al. [5], there are two primary methods of fashion virtual try-on: three-dimensional and two-dimensional try-on. Although 3D virtual try-on is excellent at showing try-on details and provides more life-like results, it is computationally intensive at the same time, which hinders its use in such applications as online shopping try-ons due to high system requirements. The GARMENTO application, however, will be designed to allow the later integration of 3D try-on if necessary, by utilizing a modular design. Sun et al. mentioned that 2D virtual try-on is the mainstream choice for the meantime [5], so our system will first demonstrate the integration of 2D image-based virtual try-on models, for simplicity.

As to 2D try-on, there has been much progress in recent years in adapting neural networks to perform virtual try-on tasks. One of the most impactful papers on Virtual Try-on neural networks is the VITON paper [6], which proposed a benchmark dataset with the same name and a two-stage coarse-to-fine try-on model. The model first generates clothing-agnostic person images, then generates raw samples using an encoder-decoder module, before further enhancements in the refinement stage. Han et al. also stated that automatic metrics like Inception Score are unsuitable for evaluating virtual try-on performance and human evaluation should be preferred [6]. Clothes warping used by VITON might leave unwanted artifacts on the final result, so later studies were carried out to counteract this issue. CP-VTON [7] made much progress in preserving body-clothes alignment and clothes details in the warping step. Its successor, CP-VTON+ [8] improved the model’s capabilities of preserving clothing shape and texture. In addition to these methods, emerging works implement Generative Adversarial Networks (GAN) (which learn to create plausible data using a two-stage pipeline: generation and discrimination) in the image synthesis stage. Yang et al. [9] adapted GAN in their ACGPN research, which claims to outperform the above methods in terms of visual quality and human evaluation when evaluated on the VITON dataset. The model, with its dedicated Clothes Warping Module, is also found to perform well even with complex posing. Also to improve clothes warping on complex poses, Minar et al. [10] proposed a 3D garment and human pose reconstruction method in which clothes fitting is estimated from a 3D model of a human and clothes. While this approach can help preserve human poses in reference images, it is far from applicable in Virtual Try-on due to the need for base 3D models for every type of clothing. As good as these models are, they are only single-garment models, which do not cover multi-garment cases where customers wish to try on both tops and pants. Neuberger et al. attempted to resolve the multi-garment problem using the Outfit Virtual Try-on (O-VITON) model [11], which is a three-stage pipeline consisting of human shape generation, followed by the coarse synthesis of clothing pieces with the calculated shape and an optimization stage based on the reference image. Also interested in dressing multiple garments are the research on DiOr [12] which also allows tucking-in and tucking-out dressing using a recurrent generation approach and the research on OVNet by Li et al. [13] which claims to be fast enough for interactive usage. Going one step further, Li et al. [14] proposed a method to allow wearing an outfit in multiple ways, e.g. having the shirt tucked in or rolled-up sleeves. These papers greatly benefit this research on exposing such capabilities in a business-to-business web application.

When it comes to integrating a virtual fashion try-on model in a web-based application, there is only one big name, Reactive Reality. Its virtual try-on product, PICTOFIT, is a fully featured application hosted by the company, offering its services to fashion brands via subscriptions [15]. The product boasts both design support and buyer try-on features in a complete web-based software package to which client companies subscribe for usage. This approach can reach and serve many businesses with similar needs; however, its customizability is limited, and the fashion brand cannot take control of the pricing when the service integrates with e-shopping websites. The integration of the service with e-commerce platforms is also limited, with Shopify being the only officially supported platform. On the other hand, developing tailor-made solutions for specific brands improves flexibility, cost control and readily available e-shop integration at the expense of development and maintenance costs. This work will attempt to create a general design of a try-on that is scalable and customizable to individual business needs.

# Problem & Motivation

This work aims to explore the development of GARMENTO, a versatile fashion try-on system that offers modularity and customization options for integration within fashion brands' existing infrastructure. Through this endeavour, we will examine the viability of adopting either microservices or modular monolithic architecture, evaluating their respective suitability. Furthermore, the design will address the significance of MLOps (Machine Learning Operations) in the ongoing enhancement and integration of the try-on deep learning model. The remaining sections are organized as follows. Chapter 2 outlines the necessary background knowledge that is required to build the GARMENTO system. Chapter 3 discusses the design of the system when viewed from different perspectives. Chapter 4 demonstrates the development of significant components of the system and possible future improvements, and Chapter 5 summarizes all the work done in this work.

1. Background Study

To successfully design and implement a modular Virtual Try-on system which can easily integrate into existing workflows, background knowledge should be presented. Firstly, the steps which the Virtual Try-on system may support are studied. Then, technical aspects are discussed briefly to reveal which technology will be applied.

# Business of Fashion Brands

Fashion is a subset of the Textile and Clothing industry, which is one of the world’s major industries. Although the industry has a long-standing history and the use of IT has greatly revolutionalised workflow of virtually every domain, the high-level process of creating a fashion product, in general, did not change much. According to Studd et al. [[16]](https://www.zotero.org/google-docs/?J6NzLw), the process for designing and manufacturing in the textile industry is an eleven-stage workflow, including Identifying needs, Specification, Studying Product relevance, Conceptual Design, Preliminary Cost Estimation, Feasibility Evaluation, Detailed Designing, Prototyping, Manufacturing, Launching and Review. Within those stages, a fashion try-on system would be the most beneficial in the Detailed Designing, Prototyping and Launching stages. In the Detailed Design and Prototyping stages, designers from the brand can enjoy an immersive try-on experience with virtual mannequins as opposed to having to assemble prototypical items only for testing out on mannequins. Producing physical prototypes runs the risk of being abandoned, which causes waste, whereas using virtual mannequins helps brands reduce the number of possible prototypes they have to produce, therefore potentially reducing prototyping costs. When the product is launched, customers can experience virtual try-on directly from online shopping sites without having to go to fashion stores only to find out whether the new clothing part suits them. This will in turn reduce the time it takes for customers to shop for clothes. Moreover, physical shops also benefit from virtual try-on when they can implement virtual reality fitting rooms to serve a greater number of shoppers without having to produce many display samples. Due to the points mentioned earlier, we will suggest a reference workflow for our GARMENTO application in Chapter 3 where we also discuss system design.

|  |  |  |
| --- | --- | --- |
| Phase | Activity | Responsibility |
| 1 | Identify needs or wants | Marketing with engineering assistance |
| 2 | Specification | Marketing and engineering |
| 3 | Relevance of product | Marketing, production, financial, legal |
| 4 | Conceptual design | Engineering |
| 5 | Preliminary cost estimation | Production |
| 6 | Evaluation | Finance and Marketing |
| 7 | Detail design | Engineering |
| 8 | Prototype | Engineering |
| 9 | Manufacture | Production |
| 10 | Product launch | Marketing |
| 11 | Product review | Marketing, engineering and financial |

Figure 1. The design process in the Textile & Clothing industry.

Taken from the book Product Design and Technological Innovation: A Reader (1986) [[17]](https://www.zotero.org/google-docs/?2r8ja2) and referred by Studd et al. [[16]](https://www.zotero.org/google-docs/?tCGCi3)

# Modular Software Architecture

A system that can integrate into an existing system with ease does not have to be highly modular; however, careful modularization may help improve system scalability since each component can be scaled if required. Traditional web-based software development usually prefers a monolith architecture where all system components reside in a single application unit. This approach, while making it easy to maintain the whole platform in a single codebase, is not easily scalable and does not accept new functions easily. There has been much effort to improve the scalability and extensibility of the monolithic approach over the years. One famous architecture that emerged in the early 2010s is the microservice architecture, which is an extension of the popular Service-Oriented Architecture. Microservices are small, single-responsibility services that work together to form a complete system [18]. Microservice architecture is often associated with distributed systems due to having multiple sub-services communicating via network messaging [19]. The benefits of embracing microservices include independent scalability, independent deployability, resilience and technology heterogeneity. Such advantages are helpful to enterprises; however, it takes much more effort to set up a complete microservice-based system than a traditional monolith due to the complexity of setting up the infrastructure. For smaller businesses, the cost of adopting microservices may outweigh all the benefits. In this case, a more recently coined term, “modular monolith” will work better. A modular monolithic application is a monolithic one, with all necessary services residing in the same codebase and deployed as a single application [20]. The benefits include cheap communication since inter-module communication is done intra-application and ease of development and integration since all modules share the same technology stack. However, such an application can outgrow the architecture to the point where redeployment for a simple change requires a lengthy and risky total system rebuild. Nevertheless, in earlier development stages where the application is still small, modular monolith architecture should be adopted to reduce time-to-market. Later on, a clean modular application can be easily refactored into a microservice-based application. In the design of the GARMENTO application, a service decomposition will be presented. Then, in the sample implementation section, we will provide a microservice-based implementation as a proof-of-concept work.

Machine Learning and Deep Learning models are essentially groups of parameters that are used together to calculate an inference result. However, there are many frameworks to develop Machine Learning models in the market, each comes with a distinct set of pros and cons and a different way of serving models. For instance, Keras does not require the presence of model definition when performing inference whereas Pytorch requires it. Fortunately, a unified format for deep learning models, ONNX (Open Neural Network Extension) was released, making it easier to store and retrieve models [[21]](https://www.zotero.org/google-docs/?OvwUb5). In addition to that, the ONNX format can fit into existing pipelines seamlessly due to rich support for machine learning frameworks and platforms [[22]](https://www.zotero.org/google-docs/?JcpdfL), making it possible to choose between a compact backend (e.g. ONNX Runtime by Microsoft) or a dedicated machine learning model service like Azure Cognitive Service without giving away hardware acceleration capabilities.

# ML Models in Production

Model storage, retrieval and inference are only a part of the development and deployment lifecycle of machine learning models. In recent years, the term MLOps (machine learning operations) has just emerged and gained much attention in the same way DevOps did previously. Some observers even put MLOps as a “competitive business value” which differentiates an enterprise from others in the same domain [[23]](https://www.zotero.org/google-docs/?hHFBCN). According to the book Practical MLOps, MLOps extends from DevOps and adds automation of Data and Platform automation [[24]](https://www.zotero.org/google-docs/?7ckxz4). While the authors claim that MLOps must be closely linked with cloud-nativeness, it is also possible to perform MLOps for on-premise systems. Currently, several end-to-end solutions on the market satisfy these requirements, for instance, MLFlow [[25]](https://www.zotero.org/google-docs/?2hnat5), Kubeflow [[26]](https://www.zotero.org/google-docs/?CmWAUN), Valohai [[27]](https://www.zotero.org/google-docs/?Aud9uo) and many more. These frameworks and platforms are also capable of exposing machine learning models via HTTP endpoints. Due to the complexity of setting up off-the-shelf or open-source end-to-end MLOps solutions and the goal of this research is creating a modular system, we will not dive into a specific MLOps framework but rather will provide a demo application using a barebone microservice to serve the machine learning model.

1. System Design

Before implementing a version of GARMENTO, a system design has to be available since GARMENTO is purposed as a guiding implementation that businesses can refer to and extend to better fit their special requirements. The main business flow will be suggested to facilitate the creation of design diagrams. Higher-level diagrams of the C4 model [28] will be presented as design diagrams since the C4 model is a well-known and widely adopted format of object-oriented system designs.

# Business Flow

First of all, it is important to come up with a business workflow that the system will fulfil to fully realize the result. According to the business discussed in Chapter 2.1, the GARMENTO system will support in Design, Prototyping and Launching phases. In the Design phase, in order not to disrupt the design flow, garment designers would continue to design using their favourite software suites, as long as the result is in image format. Designers only start their workflow on GARMENTO by submitting their work (“Create” step), and then they can fit their new design on pre-defined mannequins or reference images (“Fit” step). Upon fitting on virtual mannequins, designers can choose whether or not to keep or discard the design before asking the engineering team to produce wearable prototypes and submitting their model for managerial review (“Review” step). Proof of wearable prototypes can be attached to the design review request as images or videos. After the design is approved, it is passed through to the production department for manufacturing, which is the same as in the reference workflow presented in Figure 1 [[16]](https://www.zotero.org/google-docs/?ZLXPSD). While production is taking place, the design can be published (“Publish” step) for public try-on on e-commerce websites to facilitate online pre-ordering (“Customer try-on” step). In the Customer try-on step, potential buyers can also enjoy size recommendations based on body measurements. The complete flow is illustrated in Figure 2.

Figure 2. GARMENTO system main flow

Compared to the reference flow, GARMENTO’s main flow keeps the same order as the reference flow, which helps company staff get started with using GARMENTO. Moreover, the Customer try-on step can be integrated into online shopping sites with ease and customers can see and try-on new products virtually when they are launched, thereby improving the e-shopping experience as a whole.

# Design Diagrams

A software system can be thought of as “a hierarchy of building blocks” [29] which includes containers, components, code and the people who use the resulting software product. Brown proposed the C4 model in his book “Software Architecture for Developers” to encapsulate different levels of detail when visualizing a software architecture. By his definition, there are four levels of detail in visualizing a system: Context which views software systems in their relevance with the outside world, Container which views a system as a collection of independent deployable units, Component which zooms in further to show groups of functionality within the same deployment unit and Code which shows abstract programming language elements. In Brown’s definitions, microservices are part of the Context view if they exist outside the system, or part of the Container view if they are parts of the system to be built. This section will discuss the GARMENTO system at Context, Container and Component levels of detail without diving too much into the code granularity since our components will be designed to be platform-agnostic.

## Context Diagram

The Context Diagram reveals the role of the system to the outside world and other systems. GARMENTO is a virtual try-on platform for a fashion company so it can also integrate with existing systems that the company possesses. Since GARMENTO is designed to be generic and adaptable to various business cases, the Context Diagram will only show the virtual try-on system and its relationship with systems that are widely available among garment development companies. Figure 3 shows the System Context diagram for the Virtual Try-on System.

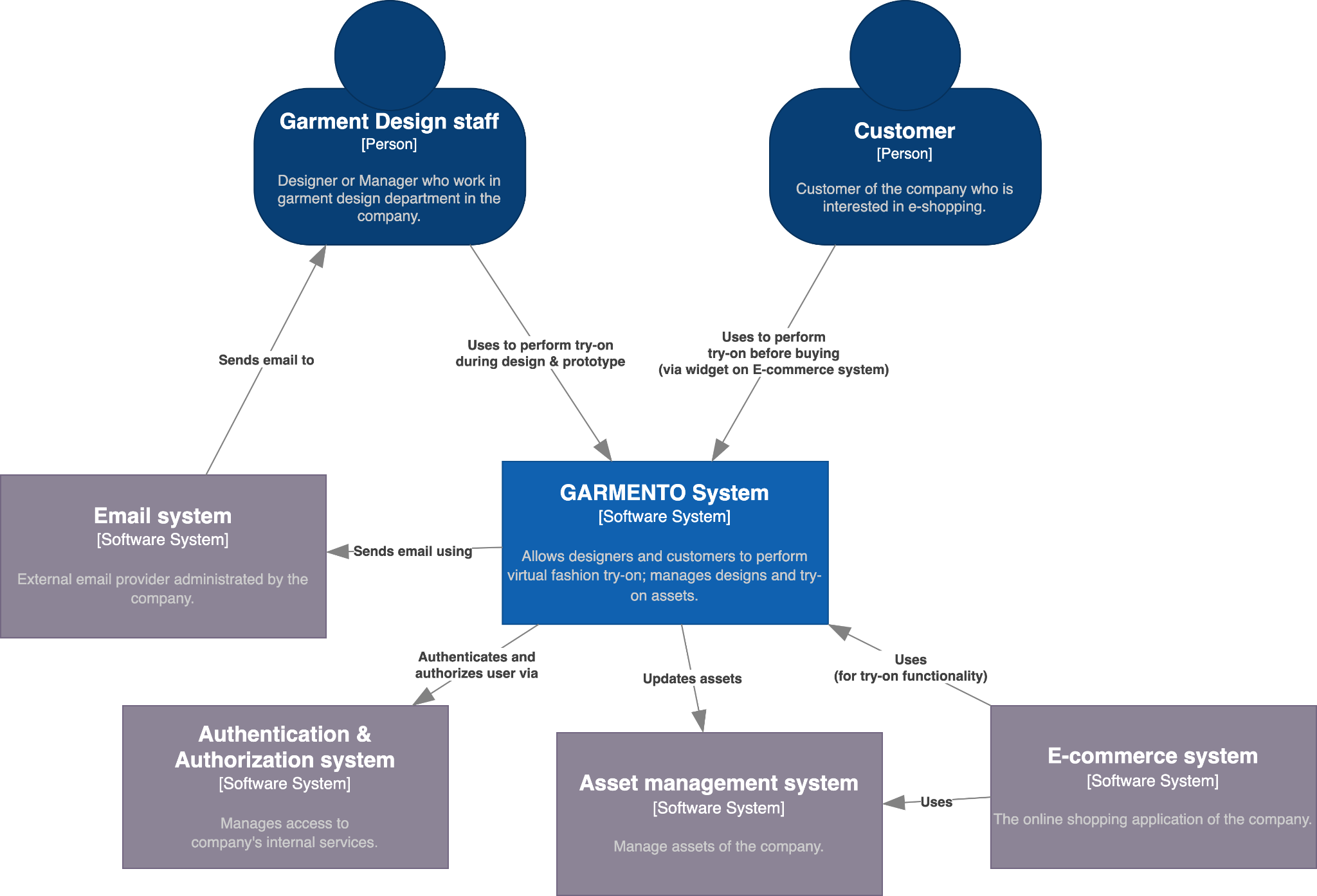
**

Figure 3. Context Diagram of the GARMENTO Virtual Try-on system.

To put into context, the GARMENTO system will be used primarily by design staff including designers to aid in designing and prototyping. As discussed in previous chapters, managers can approve and publish the design and try-on results to the Asset Management system, which is mandatory in virtually any business. To notify the manager to review a design, GARMENTO calls the External Email system, which is also available and administered by the company. Another usage of GARMENTO is to provide virtual try-on to the company’s customers via the company’s E-commerce system, which is also a widely adapted system among businesses in the garment development sector. The e-shopping system can take product assets from the Asset Management system to serve as product detail images and link virtual try-on of a product to its product details page. This point requires some integration effort to integrate the E-commerce system with GARMENTO, but the try-on system can be designed so that integration to the e-commerce site is as simple as attaching a widget to a page. Finally, GARMENTO checks for use eligibility via a dedicated sign-on system, which is mandatory for almost every business to perform access control to business services.

## Container Diagram

The Context Diagram demonstrates the GARMENTO system in its relationships with its users and other systems. To take a step further towards implementation, high-level technology decisions have to be made, as illustrated by the Container diagram. Such elements, although required to be associated with a certain platform by the diagram, can be implemented using alternative technologies, therefore they are suitable to include in this study. Figure 4 demonstrates high-level technological choices for the implementation of the GARMENTO system.

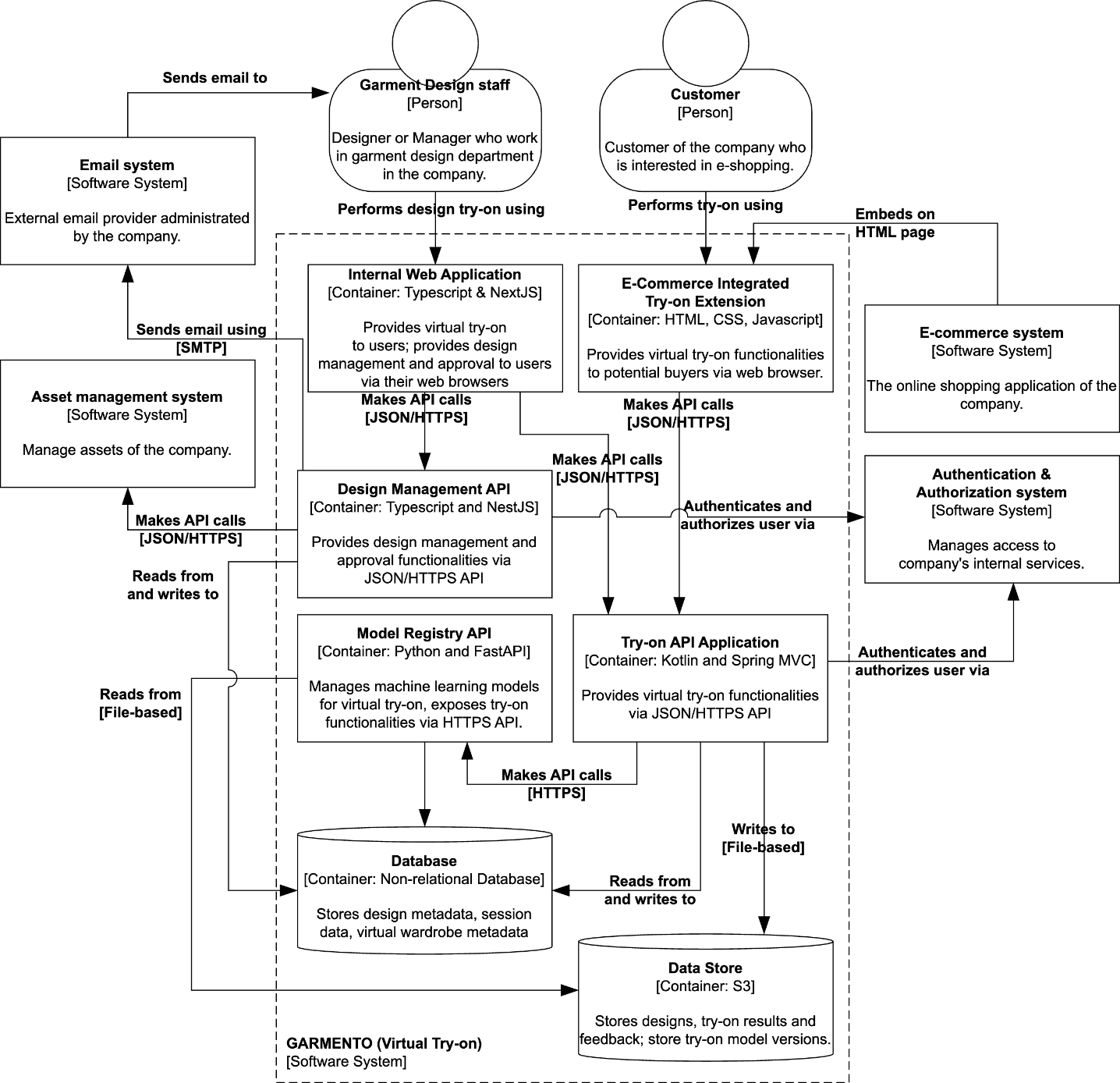


Figure 4. The Container Diagram of the GARMENTO Virtual Try-on System.

Zooming in, the GARMENTO system is composed of several inter-connected deployments. First of all, for internal users to access the system, an Internal web application must be available. This deployment unit is executed from the user’s computer and is responsible for presenting the user interface for interacting with the Try-on system. Here, the development language and framework chosen are Typescript and NextJS respectively, but in other constrained conditions, a different frontend set of development tools will also handle the job fine. In addition to the UI for internal users, GARMENTO also offers a lightweight try-on web page to be embedded into the company’s existing e-shopping application.

As the user (both staff member and customer) interacts with the frontend application, the unit calls the Try-on API application in the background. The Try-on API application is a backend application which communicates via HTTP so any language-library combination that supports the construction of HTTP API can be used, not restricted to Kotlin and Spring MVC framework as shown in the diagram. According to the application workflow proposed in Section 1 of this chapter, the system will support a Create -> Fit -> Review -> Publish -> Customer try-on workflow. To support this workflow, we define a Design Management API deployment unit which is in charge of managing designs and performing approval workflow. The Review stage requires managerial approval so the Design Management API deployment unit needs to reach out to the Email system to send notifications to the manager via SMTP, a standard interface for email transportation. After a design passes review, it is published to the Asset Management System via JSON over HTTP communication. To ensure a user is eligible to use the try-on system, the API applications are responsible for reaching out to the company’s Authentication & Authorization system in exchange for an access token. To perform inference and visualize try-on results, the try-on API application reaches out to the Model Registry API application, which manages machine learning models for virtual try-on and exposes the functionalities via HTTPS.

As to data storage, there is a Database container to store metadata related to design, sessions and virtual wardrobe. An S3 bucket data store is responsible for storing design, try-on results with feedback and model versions.

To summarize the container diagram, the Try-On API Application deployment unit is the heart of the GARMENTO application, therefore, the next section will discuss the component diagram for this container. The Model Registry API, on the other hand, is also an extremely useful container for this system; however, it can be easily replaced by end-to-end solutions such as MLFlow, Kubeflow, Valohai, etc., therefore, its details will not be discussed as thoroughly.

## Component Diagram

A Component Diagram shows finer-grained details of previously mentioned containers. According to Brown, the Container diagram is a useful diagram to demonstrate the decomposition of a container into implementation-ready components with distinct responsibilities [[29]](https://www.zotero.org/google-docs/?qytb06). The diagram, while making much effort to maintain its up-to-date status, can be helpful when structuring the codebase. This diagram is especially valuable when dealing with monolithic applications since internal components are not explicitly displayed by the Container diagram, whereas microservices are single-responsibility components by definition so no more detail needs to be further visualised. In this case, the Try-On API Application and the Design Management API are candidates for component decomposition. The Model Registry API container is deliberately omitted from the component diagram, since in production, well-known MLOps platforms such as MLFlow or Kubeflow will be used instead. Most MLOps toolsets include HTTP endpoints for inference and model and data versioning and management, which already fulfils the functions required by the Model Registry API deployment unit. The Database and Data Store containers also do not need to be further decomposed since they are already offered as deployment units.

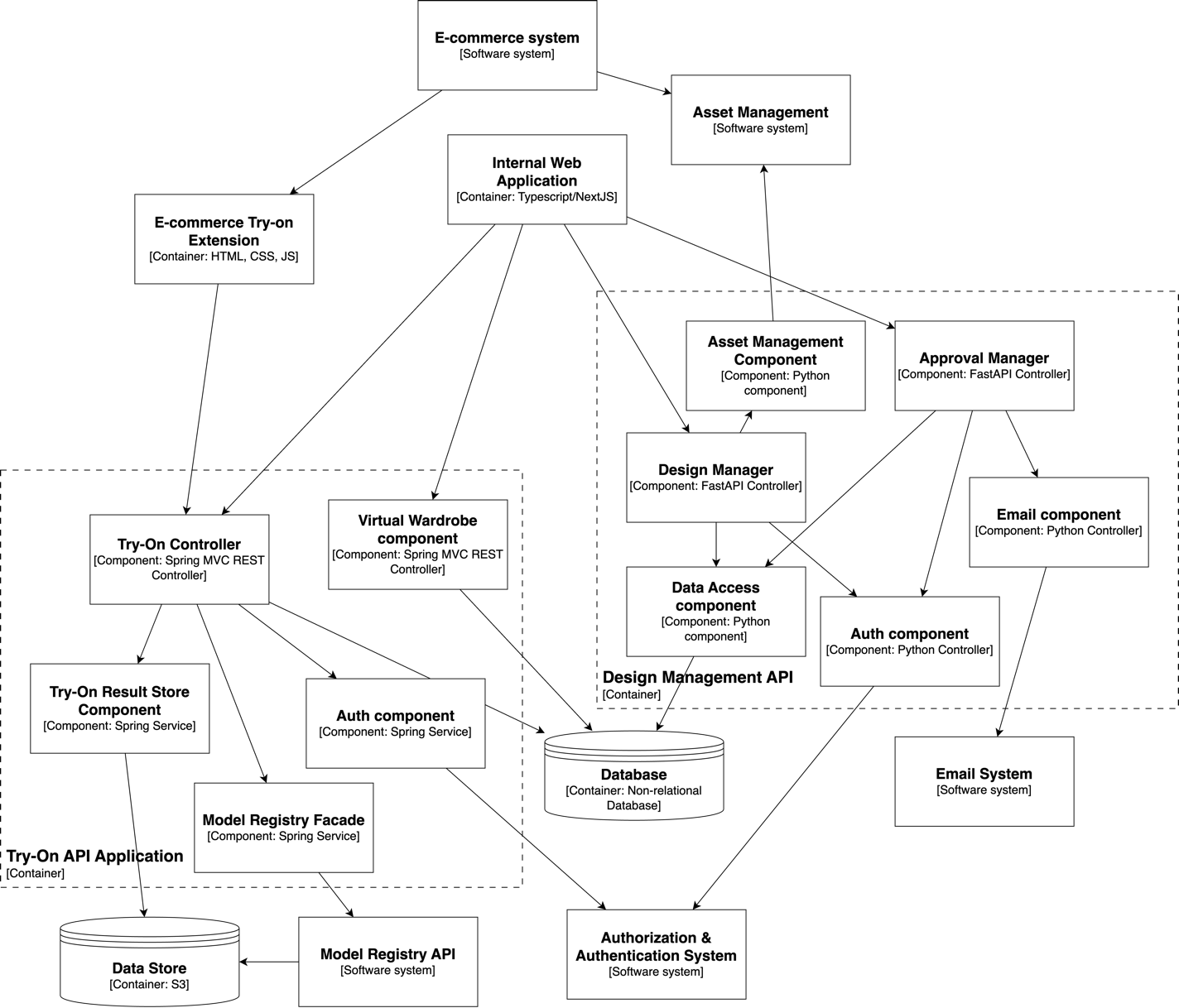


Figure 5. The Component Diagram of GARMENTO Virtual Try-on System.

Regarding the Try-On API Application container, there are several decomposed components. The first one is the Try-On Controller which is the gateway for virtual try-on functionalities. To verify a user’s eligibility to use try-on features, there is an Auth Component which handles authorization with the Authentication & Authorization system. The user will be expected to use the key issued by the Authentication & Authorization system to use the Try-On API Application. To get the result using the virtual try-on machine learning model, this controller needs to request the Model Registry API via a Facade, which is shown here as a Model Registry Facade. The result store component is shown explicitly here as a Try-On Result Component. In addition to the Try-On Controller, there is a Virtual Wardrobe Component which provides virtual wardrobe functionalities via a REST controller.

As to the Design Management API container, there are controllers for the Approval Manager and Design Manager to provide respecting functionalities. There is also an Auth Component which does the same thing as the one found within the Try-On API Application scope but can be implemented using a different technology stack. The Approval Manager component is also responsible for notifying the manager to add design approval so it needs to communicate with the Email System via an Email Component.

To sum up the Component diagram, it not only shows the outbound components (Controllers in this case) but also demonstrates the integration of containers with external systems (mostly outlined as Facade in the above diagram). The next section will discuss the transformation of the above diagrams into code and how our code can align with architectural elements.

1. Implementation & Discussion

This section will discuss the prototypical implementation of the GARMENTO system, which omitted non-critical functionalities such as email sending, state-of-the-art machine learning model management and so on. Only core try-on features, simple cataloguing, plugin-based integration and simple model management are considered in this section. The deployment using Docker Compose [30] will also be presented to demonstrate independence of deployment, however, in production builds, it is recommended to deploy the system using a cluster-based orchestration approach such as using Kubernetes [31].

# Try-On & Catalog Module

In this prototype of the GARMENTO system, a try-on is carried out by synthesizing from a reference image and a garment image created by the designer. On the designer side, preset reference images which are preprocessed for the deep learning model are offered in a dropdown. On the customer side, there might be the desire to apply try-on with custom images, however, for demonstration purposes, using preset reference images is sufficient. The following figure demonstrates GARMENTO’s try-on UI.

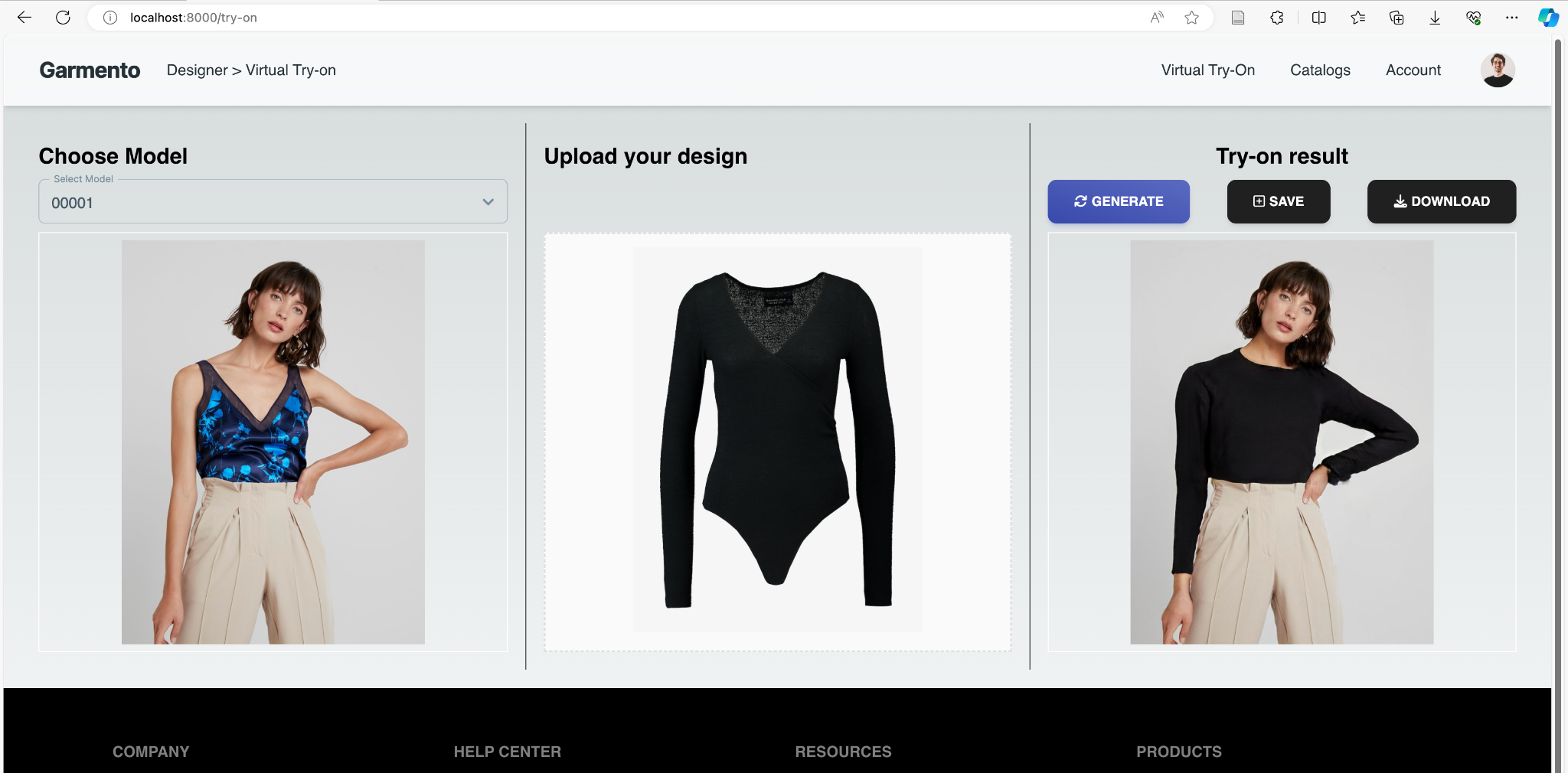


Figure 6. User Interface of Try-On function.

At the bare minimum, three columns show three steps of the virtual try-on process. The first column contains a dropdown of preset reference images which the designer can choose from. The middle column is where the designer drags and drops the design image, and the rightmost column shows the generated result and additional actions such as Save to Catalog or Download. The user interface structure is fully customizable as long as the UI still calls the correct APIs, allowing for much flexibility when developing the production version of the application. Behind the scenes, several processes are going on. Before try-on is performed, the designer has to identify himself by logging in via an authentication service. Many companies allow sign-on using Google Workspace emails with their domain, therefore, the prototype version of the GARMENTO application will also use Google-based login. The service-level access token exchange process is illustrated in the following sequence diagram.

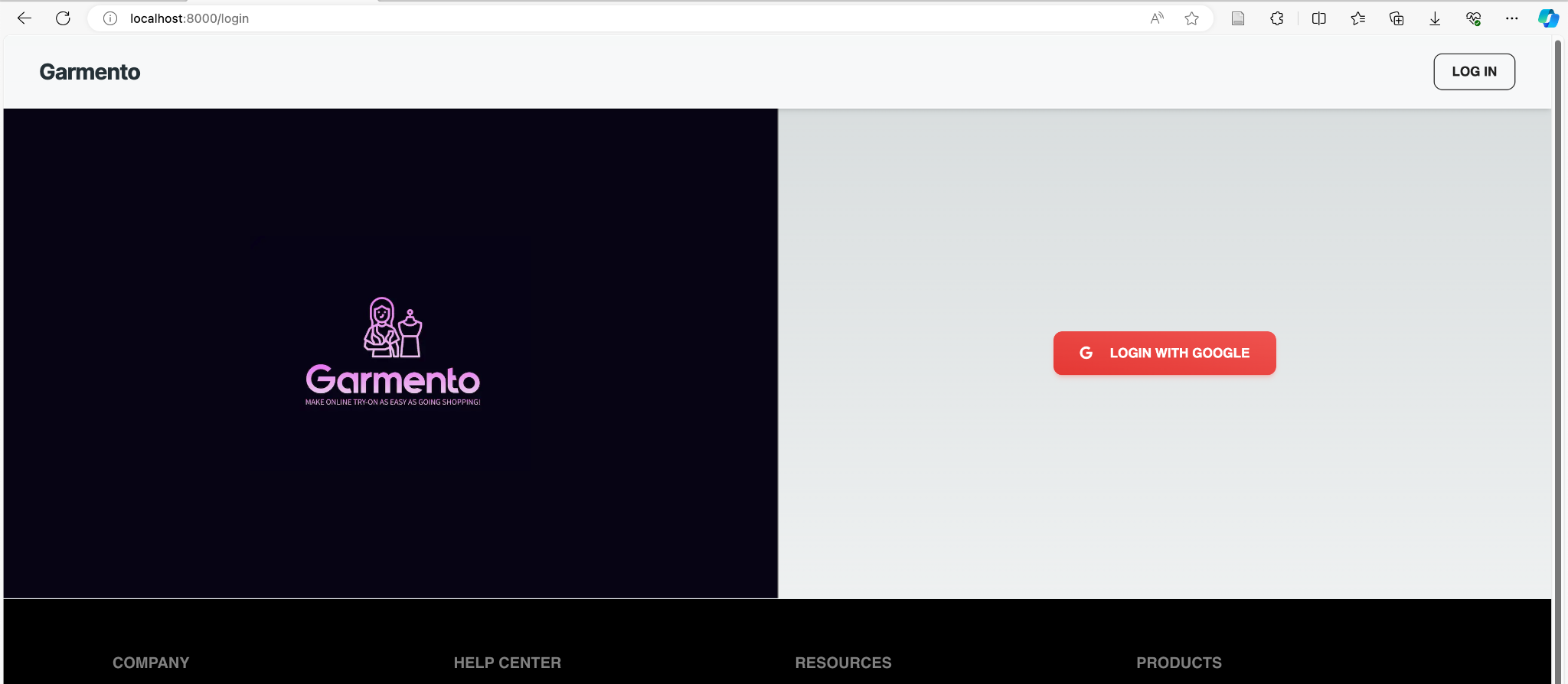


Figure 7. Google sign-on screen.

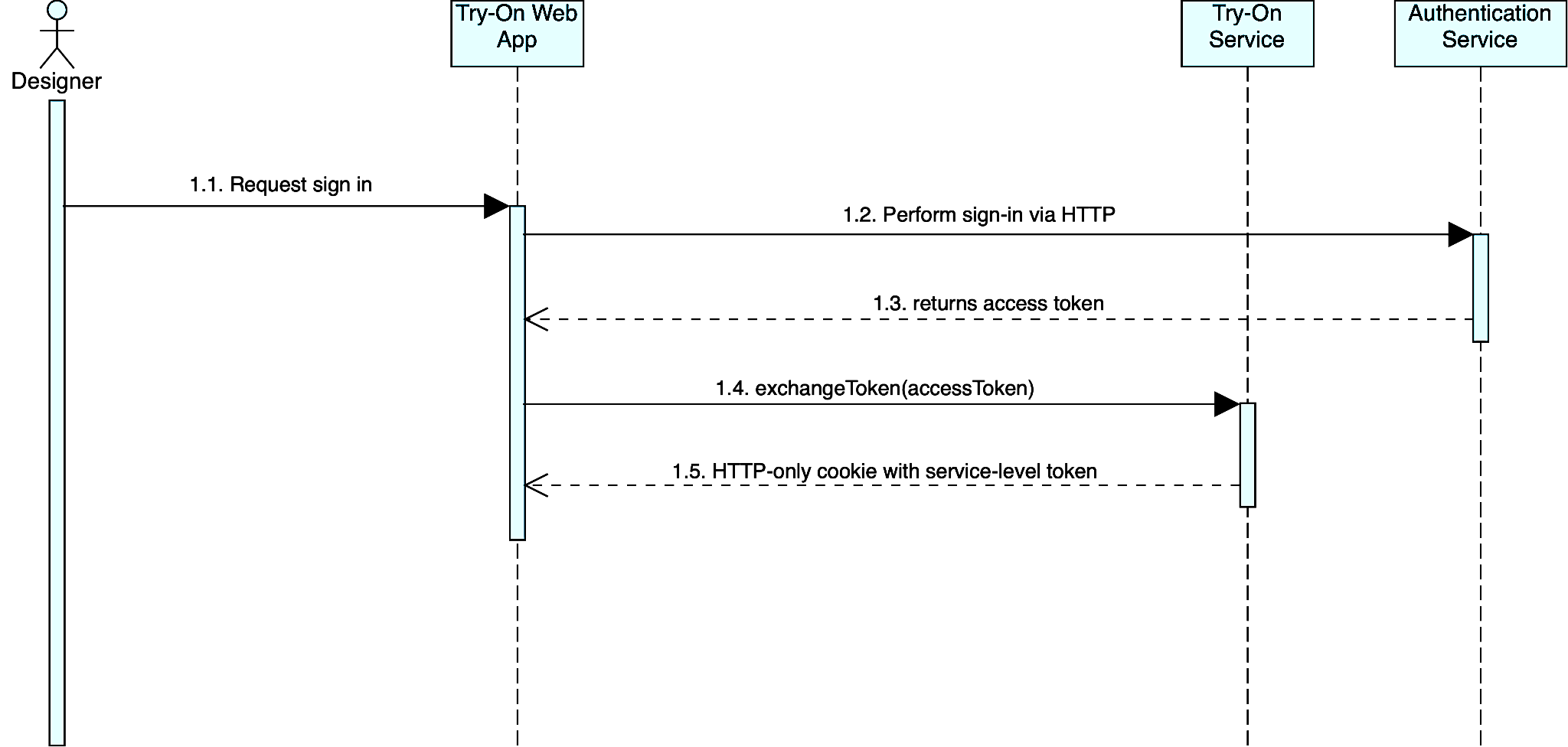


Figure 8. Obtaining service-level credentials for the Try-On & Catalog module.

Earlier in the process, the user is presented with a Login with Google button. Upon clicking, the user is taken to the Google login screen; after Google sign-on, the user is given a Google access token with minimal scopes. This access token is then sent to the Try-On service, in which the privileges of the user owning the token are queried. A JWT token is then issued if the user is allowed to access GARMENTO and sent to the client via HTTP-only cookies that are unreadable from the client's Javascript code. Storing the access token as an HTTP-only cookie removes the need to attach the token to request headers and guarantees that the token is always successfully sent to the server. It is also more secure than the other well-known methods of storing access tokens, localStorage and sessionStorage storing since cross-site scripting is no longer an issue when client code cannot read access tokens. Afterwards, every request to the Try-On service will implicitly include the access token.

When the user successfully identifies himself, the next step is to perform a virtual try-on. The full process is demonstrated in the following sequence diagram:

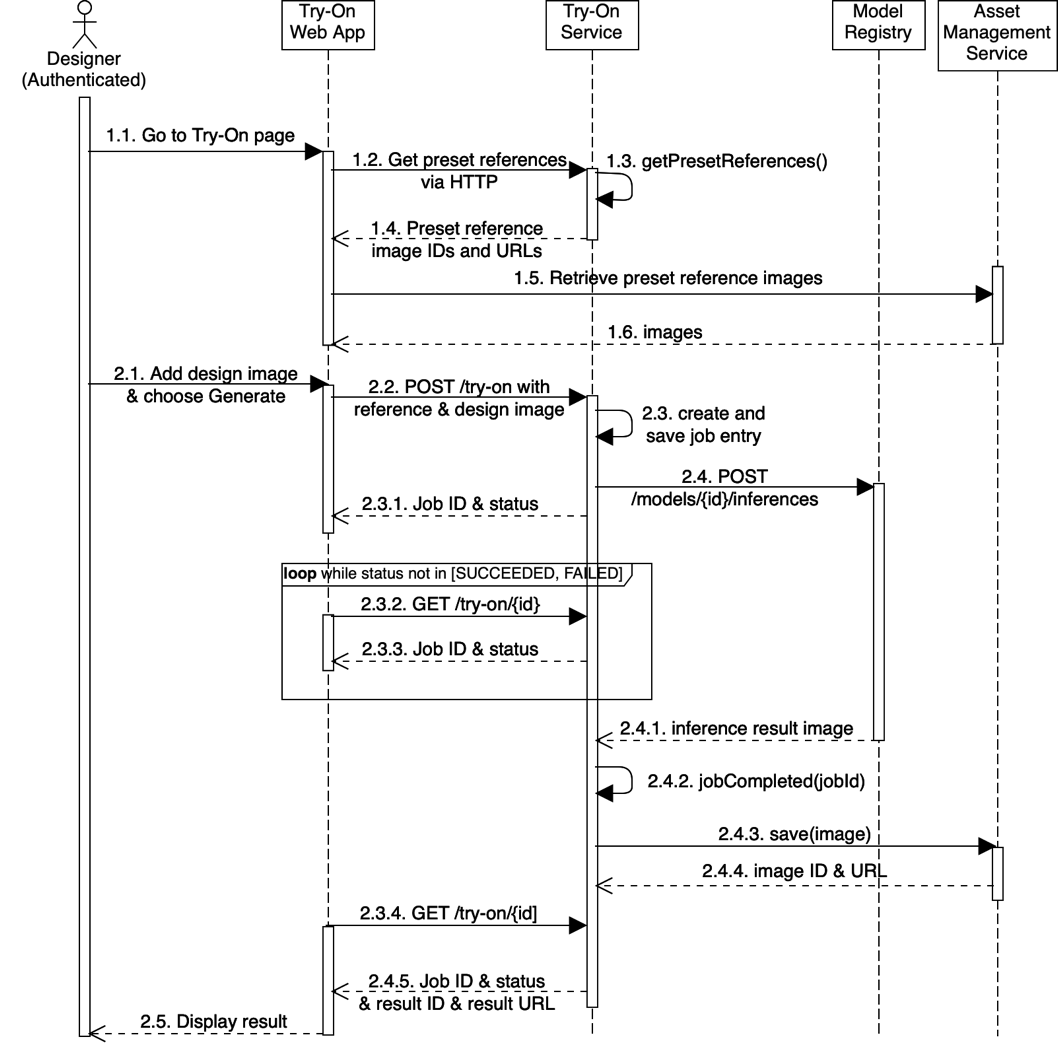


Figure 9. Performing virtual try-on with preset.

When authenticated as a designer, when the user goes to the Try-On page, the application calls the Try-On web service to get reference presets, which are simply assets managed by the Asset Management service with an Identifier and a public URL. Upon getting the reference presets, the client will display them by reaching the public URLs referring to the Asset Management service, returned in the Try-On service’s response. Then the user can select a reference image and browse a design image from his computer, before clicking on Generate to request the Try-On service to start working on try-on generation. To allow for better request handling, when successfully acknowledged try-on generation requests, the Try-On service creates a job entry for the generation task and returns the job identifier to the client immediately. In the background, the Try-On service is responsible for calling the Model Registry to predict the result from the reference image and the design image (or garment image). A background thread of the Try-On service waits for the completion of try-on synthesis, then updates the try-on job status from IN\_PROGRESS to SUCCEEDED or FAILED. During the process, the front-end application repeatedly polls the Try-On service for try-on job status (also known as short polling). Finally, when the job status is transitioned to SUCCEEDED or FAILED, the front-end app either renders the resulting image or shows the error message.

A catalogue is a collection of designs which are available for public try-on after being approved. Each design in a catalogue is accompanied by one virtual try-on result to make it easy for managers to approve. In the prototype, a catalogue can be in either of these statuses: Draft (default when created), Submitted (ready for review), Approved, or Published. State transition between these statuses is illustrated in the following figure:

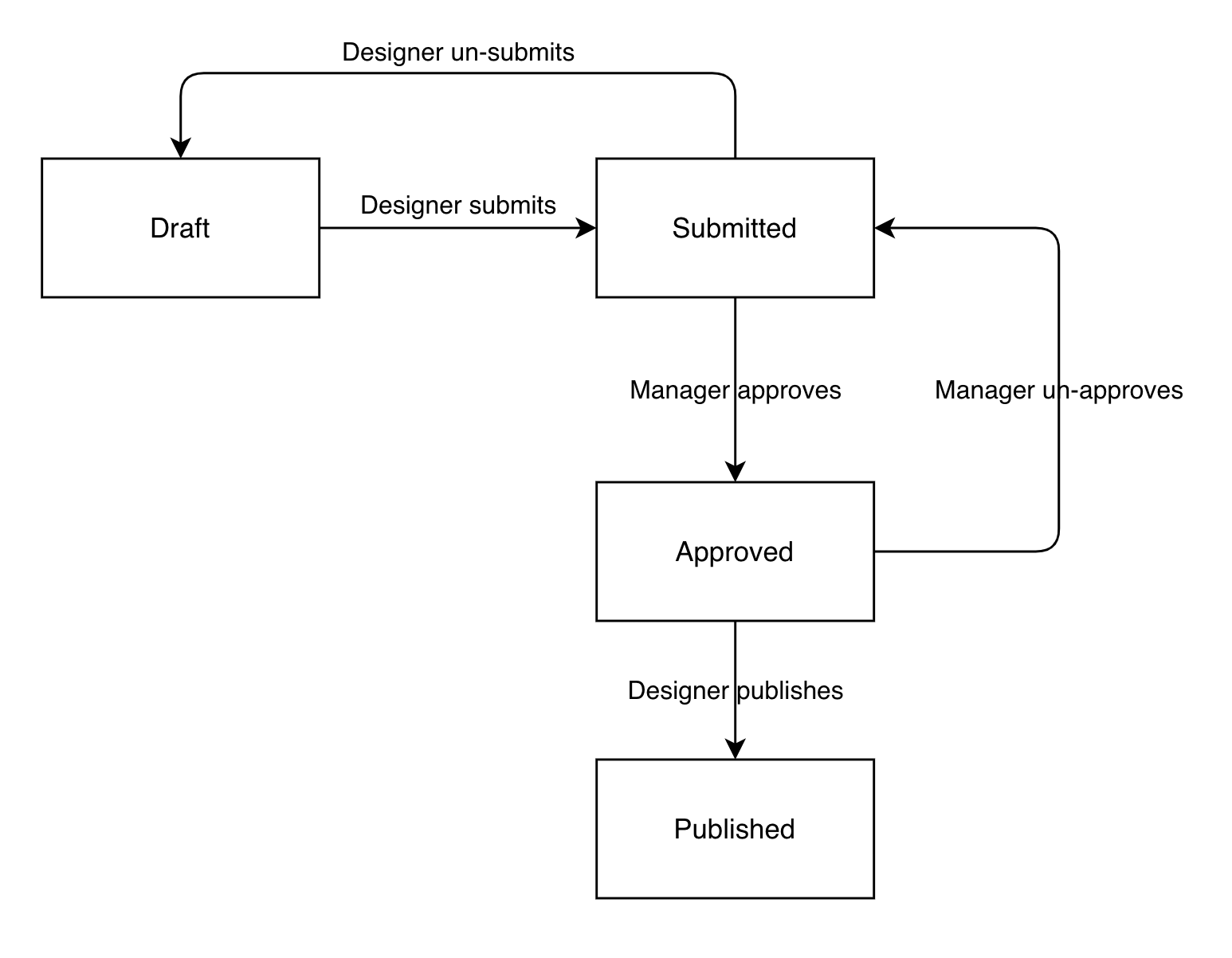


Figure 10. Catalogue status transitions.

When in the Draft status, the designer can freely add more items to the catalogue. However, when the state becomes Submitted, no more items can be added to the catalogue. If a catalogue needs revising, the designer has to un-submit it first. Then, after the catalogue does not need any more edits, the manager can approve it so that the catalogue becomes Approved. Only in the Approved state can the catalogue be published, and designs in the catalogue become publicly accessible in the customer try-on widget on an e-commerce application.

# Try-on Integration into an E-commerce Application

Integration of try-on services to a public application makes the exposed services prone to misuse. Since requests to the e-commerce application neither need to be authenticated nor authorized, abusers can take advantage and launch denial-of-service attacks which can result in crashing the model server (which is the most computationally demanding component of this application). As a result, only publishing try-on services to public endpoints can lead to potential security issues and should not be applied. A promising approach is to wrap the necessary components for virtually trying on a specific design into an iframe component and embed it into the e-commerce site. This approach allows for easier integration since only one additional component needs to be added to the e-commerce site and also enables advanced security control measures to be applied, such as X-Frame-Options and Content-Security-Policy headers. Moreover, embedding a try-on plugin into the e-commerce website allows the plugin to conveniently update without making modifications to the host website. That means a 3D try-on service, when available to use, can be integrated to the e-commerce website only by deployment to the try-on services. Finally, rate limiting can be enforced only on the plugin without having to make changes to the host application. The following figure demonstrates how such a plugin fits into a product details page.

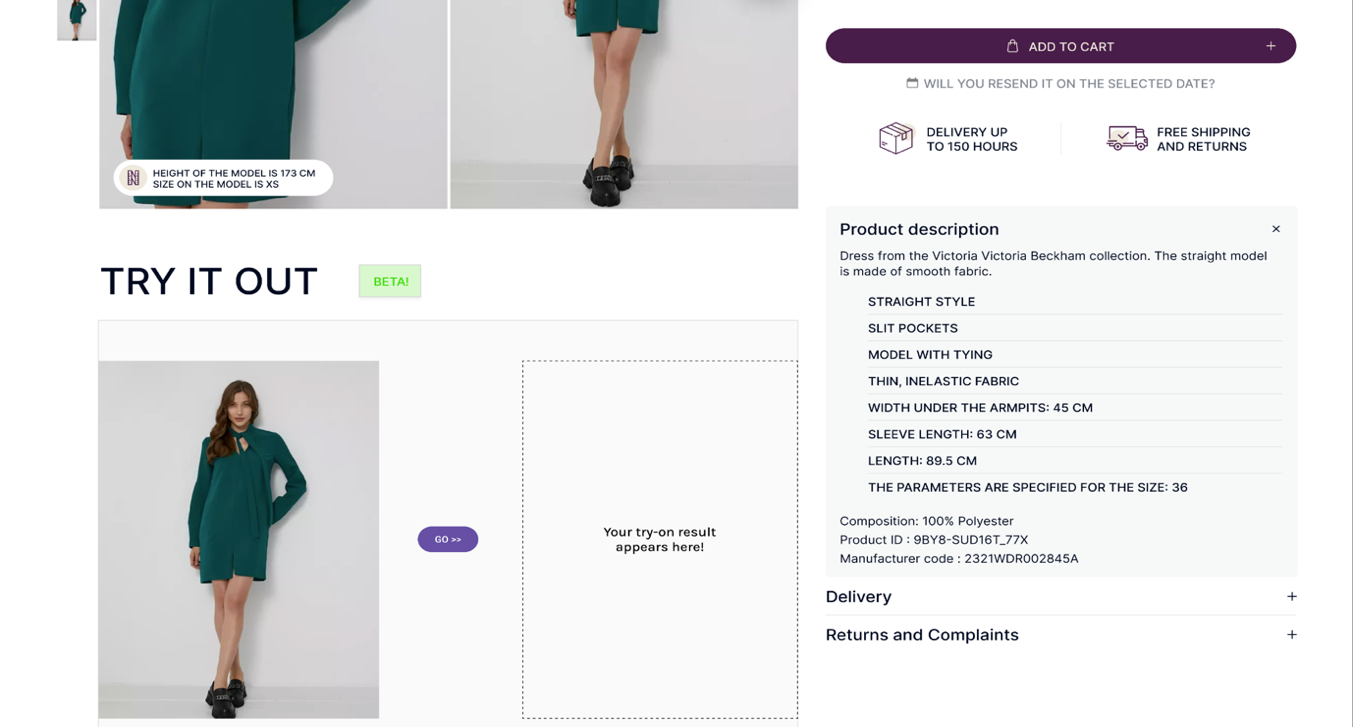


Figure 11. Virtual try-on plugin integrated into an example e-commerce web application.

The above image shows one way that the try-on function plugin integrates into an existing application. The “Try it out” section is a small section that is loaded from the Try-On web application, which has a compact two-column layout, one holds the customer’s uploaded image for trying on, and the other shows the result after applying a virtual try-on. The “Go” button in the middle initiates the try-on process. The section does not include an input field for the design; instead, the product identifier is fed to the widget and the widget is responsible for collecting the design metadata from the try-on service. When the customer clicks on the “Go” button. The following sequence diagram illustrates how the plugin works behind the scenes.

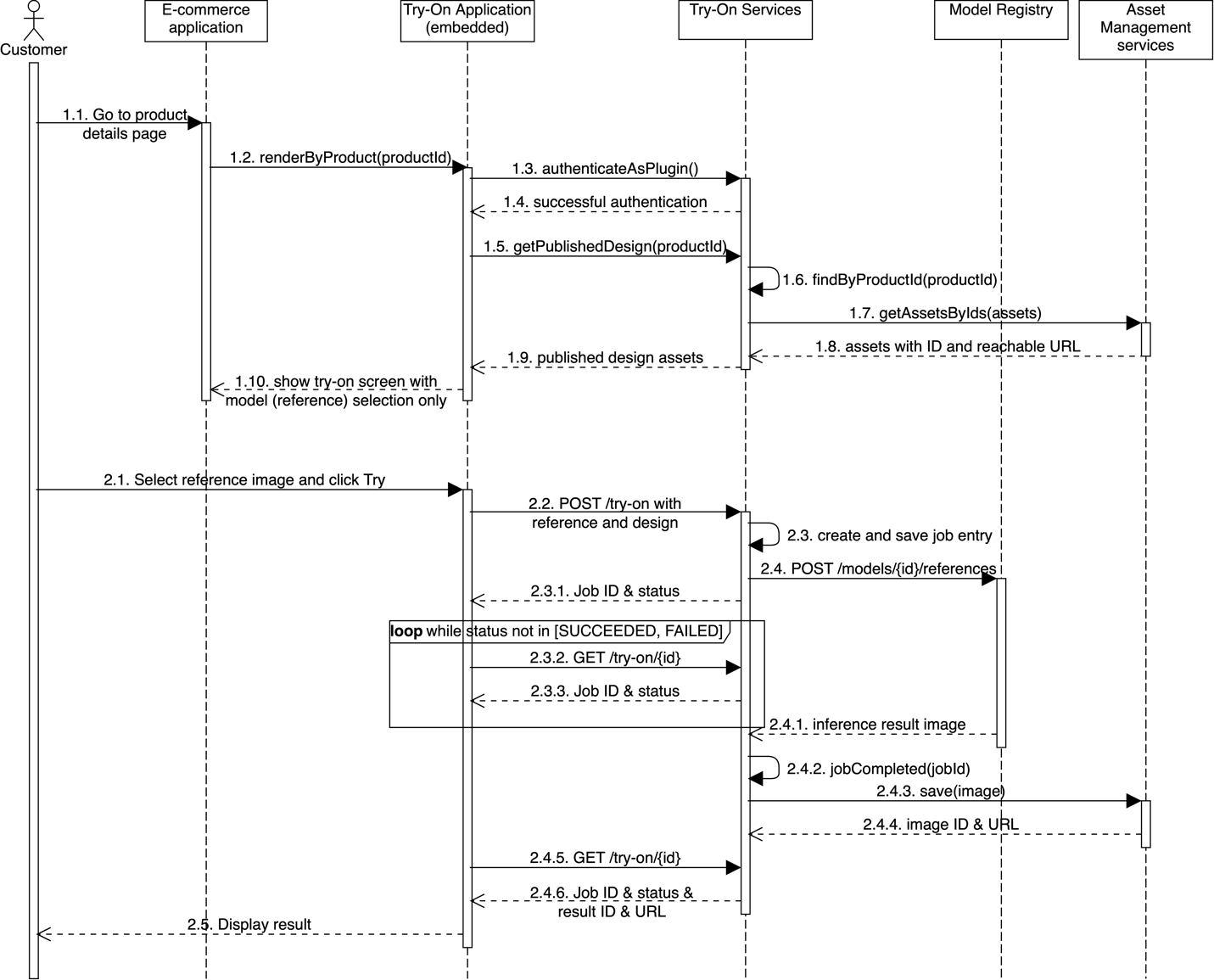


Figure 12. Sequence diagram for E-commerce Application integration.

When the user goes to the product details page, the page gets rendered, which also triggers the render of the try-on customer widget if the current product has its design published for try-on. On initialization, the widget (shown on the diagram as a Try-On Application (Embedded)) authenticates itself as a plugin against the try-on services using server-side authentication to ensure that the widget is loaded from a trusted site by examining the “Referrer” header. When authentication succeeds, the widget retrieves and stores the metadata of the corresponding design in the local state and shows a try-on screen as shown in Figure 11. When the customer uploads his reference image and clicks on Try, the widget performs a similar process to that of the designer-side try-on discussed in Chapter 4.1 and displays the result in the right column.

# MLOps Implementation

There are numerous tools to support MLOps in the market as mentioned in Chapter 2. This sub-section will discuss a simple machine learning model management and exposure method using plain Python along with the Lightning framework for easy model prediction and FastAPI for REST API exposure, without diving fully into a complete MLOps flow. GARMENTO is a web-based application, therefore, model management functionalities are offered through HTTP APIs. The sequence diagram for the model registry is presented in the following figure:

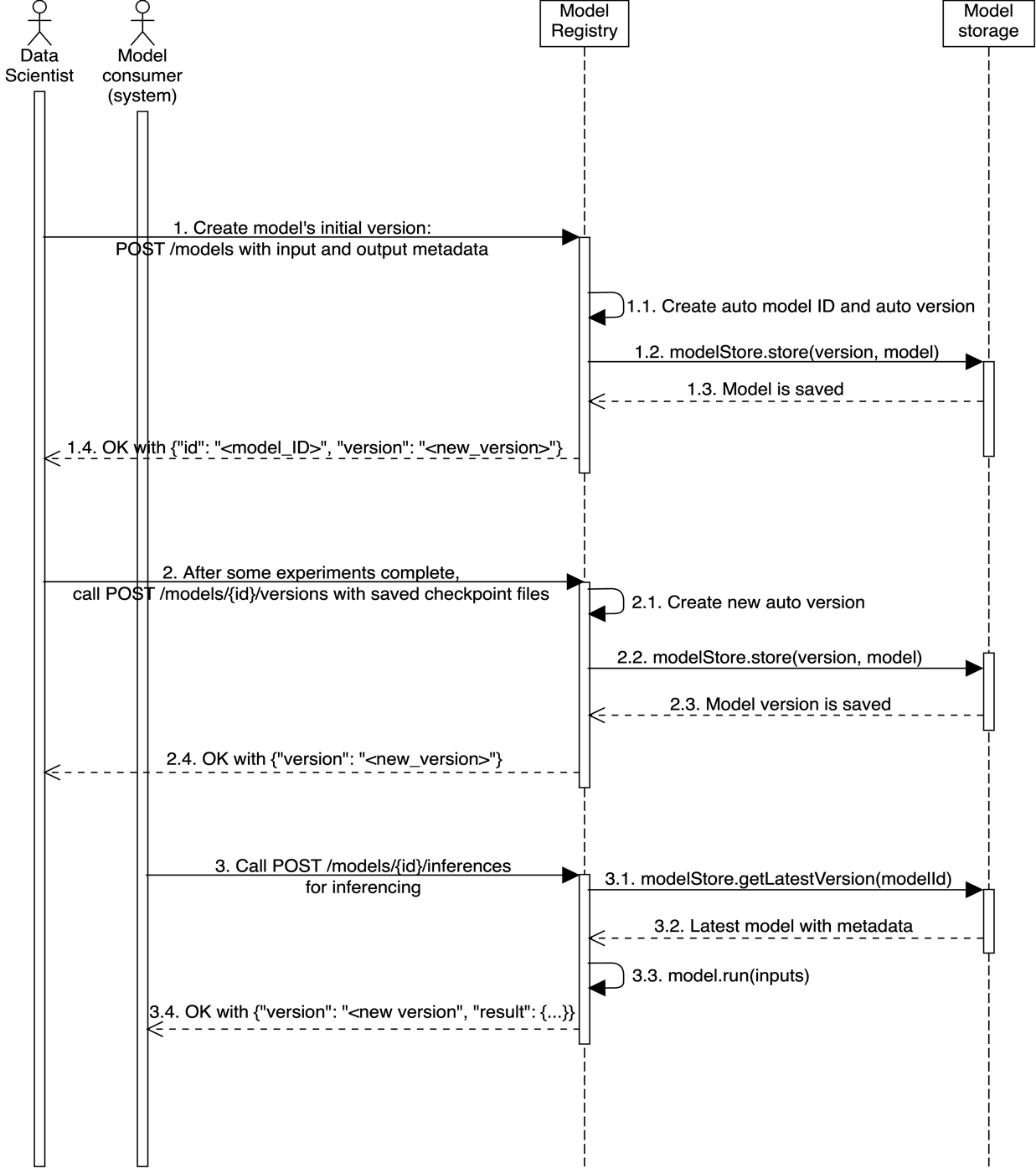


Figure 13. Model Registry’s Sequence Diagram.

First, the data scientist implements and performs basic experiments in his computational resources. When the model is good enough, he exports the model checkpoints and publishes the model to the Model Registry via a POST call to /models endpoint. The input to this operation must include input and output metadata to make it safe for inferencing. The model registry service will take care of creating an identifier for the model and an initial version. The addition of a model version is similarly done: when a model has been sufficiently further trained, the data scientist also performs the checkpoint exporting step and requests a new version using a POST call to /models/{id}/versions. This version tracking workflow is similar to many offerings in the market, so it is easy to swap out for more advanced model registry solutions such as MLFlow.

Regarding inferencing, when the model and its versions are available in the Model Registry, any client application can request the /models/{id}/inferences endpoint to utilize the model for specific applications. In reality, a machine-learning model might be saved in several different formats, which poses challenges to the model registry in adapting its operations to distinct formats such as PyTorch or TensorFlow flavours. A unified format like ONNX can be helpful for model registry systems, eliminating the need to build adapters for multiple formats and resulting in more lightweight deployments. Deployment-wise, the Model Registry is independent of other application services, therefore it can be deployed independently as a containerized unit managed by container orchestration platforms like Docker Compose or Kubernetes. When deployed to Kubernetes, the service can take advantage of advanced orchestration features such as horizontal pod auto-scaling to handle increased traffic and ensure optimal performance under varying load conditions, or health check to automatically redeploy pods to achieve high availability.

Due to the independence of the service, integration into the existing systems is easy. In the GARMENTO system, to integrate with the main Try-On service, the Model Registry needs to be discoverable by the Try-On (Designer) service. Thanks to the use of a service registry service based on Netflix’s Eureka server [[32]](https://www.zotero.org/google-docs/?Juvu67) (which will be discussed in the following section), service registration can be done with ease. To summarize this sub-section, while there are many aspects to MLOps, the aspect of model versioning and inference is the most important one. Deploying the model registry as an independent service makes it easy to integrate with the existing system while allowing for rapid scaling when deployed on suitable orchestration platforms such as Kubernetes.

# Microservice Implementation

The development of the GARMENTO prototype followed the microservice-based approach whose guides are generally available either as a book [[19]](https://www.zotero.org/google-docs/?NLqwUY) or online [[33]](https://www.zotero.org/google-docs/?Opm6c0).

* 1. **Service decomposition**

Instead of having all services grouped into a single monolithic unit, we applied the approach “Decomposition by Subdomain” presented in the Microservice.io guide [[34]](https://www.zotero.org/google-docs/?4ekvwA) since it helps stabilize the architecture and keeps services loosely coupled. Subdomain-based decomposition of GARMENTO services can be seen in the following figure:

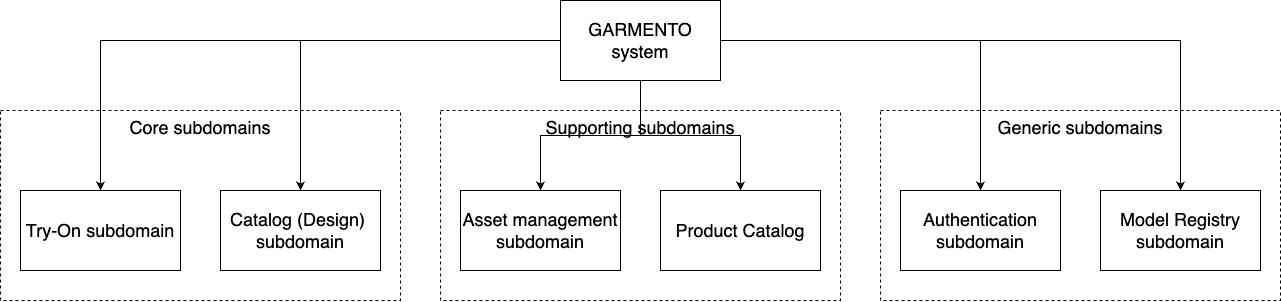


Figure 14. Service decomposition in the GARMENTO system

In its simplest form, a core subdomain of the GARMENTO system would be the Try-on subdomain in which we included the Try-On service which deals with creating try-on results, and the Catalog subdomain with the Catalog service which manages the grouping of try-on results into publishable catalogues that support customer virtual try-on upon publishing. Supporting subdomains include the Asset management subdomain which is implemented by a different team from the Try-On team, providing asset management functionalities company-wise. Generic subdomains include the Authentication subdomain which is based on Google authentication in the prototypical application and Model Registry which can be implemented by employing off-the-shelf solutions such as MLFlow, as discussed in Chapter 4.3. Another subdomain which can participate in the GARMENTO system is the Product Catalog subdomain from an existing e-commerce application. In the diagram, such a subdomain will be considered a supporting subdomain since it does not directly contribute to the success of the GARMENTO application. To further demonstrate the flexibility brought about by using service decomposition, we chose different technological stacks for each of GARMENTO’s components. For instance, while the Try-On services are implemented using Spring Boot in Kotlin language, Asset Management services are built on top of Python and the FastAPI library. The use of different programming languages promotes communicational patterns which will be discussed in the following section.

* 1. **Data Management & Communication**

In addition to service decomposition, another item we followed in the official Microservices guide is Data management and Communication. Firstly, the guide states that each service has its private database. Even in the prototype, databases for different services in the GARMENTO system are separated into different instances that are only accessible from the service deployment unit. For instance, the database used for Try-On services is an instance running MySQL as the engine, whereas currently, a PostgreSQL instance is hosting model metadata in a simple implementation of the model registry in the GARMENTO prototype. Since the Catalog services are independent of the Try-On services, its database is also split; however, since it works closely with the Try-On services, the separation is only on a conceptual level by utilizing two different databases in a single MySQL instance. Then there is the use of API Composition, with the Try-On service reaching out for Asset Management services to retrieve original assets before feeding into the Model Registry for prediction and getting the result from the Model Registry. The Catalog service also queries against the Asset Management services to retrieve catalogue items. Inter-service communication is currently handled through Remote Procedure Invocation (RPI) [[35]](https://www.zotero.org/google-docs/?Ok40rG), which is based on HTTP REST API in the latest version of the GARMENTO system. RPI requires a form of service discovery, which has to be added to the list of deployed services.

* 1. **Deployment**

Finally, the deployment of the GARMENTO prototype is highly pivoted towards official guides by Microservices.io. Several guidelines have been followed, including using API Gateway to route requests into respective services and using Client-side service discovery to allow inter-process communication. The following diagram illustrates the physical components employed in the prototype of the GARMENTO system.

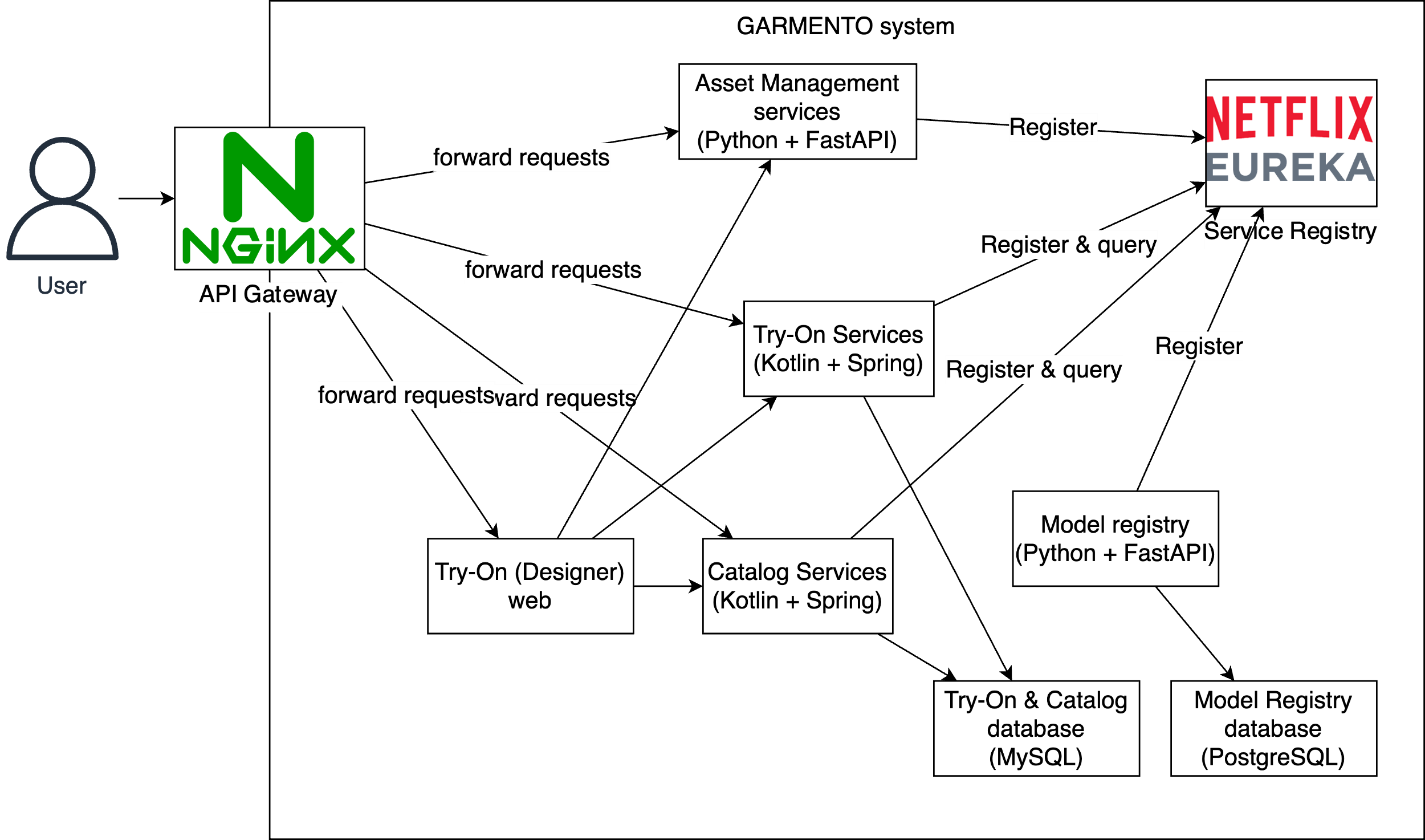


Figure 15. High-level deployment diagram (non-UML) of the GARMENTO prototype.

Each node on the above diagram is deployed inside a single Docker container. The orchestration of the GARMENTO prototype is managed by Docker Compose. An NGINX container stands at the front of the application, which plays the role of a simple API gateway, routing incoming requests to suitable service handlers. In this simple prototype, due to the small scale, there has not been a Load Balancer, but it is easy to add when the application grows quickly in size. Behind the API Gateway are service components. There is a Try-On web frontend which exposes system functionalities through HTML, CSS and JS in users’ browsers. To fulfil business actions, the front-end application is responsible for calling other services such as Try-On services, Catalog services and Asset Management services, which are deployed using different technical stacks. Try-On services and Catalog services are built using Spring framework in Kotlin language, whereas Asset Management services and Model Registry use FastAPI in Python language. To let services discover each other, a Service Registry built on top of Netflix’s Eureka service discovery tools is employed. As discussed before, the database of Try-On and Catalog services is hosted on a MySQL-based container whereas the Model Registry database is based on a PostgreSQL container. When it comes to networking, all those services share the same network to keep things simple. When orchestrated on Docker Compose, it is not necessary for inside applications to expose ports to the host machine. There is only a single access point from the API gateway, making inside components secure against unwanted access. To improve security aspects, NGINX can be swapped out for a more advanced gateway solution such as Netflix’s Zuul, which offers finer-grained control over request routing, enabling early short-circuit of suspicious requests. The current approach has allowed independent deployment of different services of the GARMENTO system; however, it has not yet supported auto instance scaling since Docker Compose is incapable of such operation. To take advantage of auto-scaling, one can resort to container orchestration on Kubernetes, which includes more advanced features in request routing and networking control.

# Discussion, Future work & Conclusion

This section will provide an analysis of the design choices, implementation challenges, and user feedback surrounding GARMENTO, our modular virtual clothes try-on system. First, our system was built using the Microservice architecture, which brought about several key advantages: high flexibility in technology stack choices which allows for Java services and Python services to work together in the same application; and independent scalability which allows services to scale up or down in the number of instances in response to different traffic conditions. When it comes to maintenance, maintaining each service is no hassle and can be done by different staff without affecting each other. However, this approach requires the set up of a dedicated development team to perform development and maintenance, which is only feasible for medium and large enterprises. Deploying microservices also requires special infrastructural tools such as Kubernetes or Docker Compose. For smaller businesses, organizing the source code following the monolithic approach is a better choice to avoid additional complexity caused by microservices deployment.

Next, there are challenges when performing migration of the GARMENTO system into the existing ecosystem of fashion brands. Not all brands operate in the same manner as shown in this work, therefore, the modules handling enterprise-facing processes might be different for different companies. Nevertheless, the core try-on function shall be relatively stable, lending itself to module reuse. The model registry component can also be based on an off-the-shelf offering. Only supporting modules such as Catalog Management, Authentication and Asset Management are company-specific and require business analysis to perform adaptation. Thus, the GARMENTO system can be nearly seamlessly integrated into enterprise-side workflows. Regarding customer-facing functionalities, incorporating GARMENTO’s customer try-on with e-commerce systems is a breeze, requiring the addition of a mere component into the existing user interface.

Another explored aspect is the use of MLOps in model improvement. Although MLOps implementation in GARMENTO only included model versioning and exposing, the impact of the tool is significant. It allows model update and continuous model integration almost instantly, requiring only an API call. Integration of a full MLOps flow might be an interesting topic for a follow-up paper. Regarding security, the current system has implemented the necessary security measures in all presented components. However, the inclusion of security methods has not lived up to current industrial standards. Thus, a later version of GARMENTO should also discuss how the system can be implemented to be compliant with GDPR (General Data Protection Regulation).

Compared to a well-known solution, PICTOFiT, GARMENTO takes a clear lead in customizability, offering as much customization as required by the business due to services being built from the ground up. While the feature set of GARMENTO has not been as rich as PICTOFiT, additional features can be iteratively added and continuously deployed. The cost of using the service is also better controlled by the fashion company without being restricted to a specific token count per month.

While GARMENTO has fulfilled foundational features for virtual try-on, there are several points for future exploration and enhancement. First of all, the current system is only based on image-based (2D) try-on. It would be a huge step forward and also pose several challenges when we experiment with the use of 3D-model-based try-on, as deep learning models for 2D and 3D try-on are vastly different and the display of 3D try-on on the Web comes with concerns about performance degradation. Along with 3D try-on, video-based try-on and better, augmented reality-based try-on will further elevate the virtual try-on market. A more in-depth view of user experience is also helpful to refine the user interface of GARMENTO, which will enhance user satisfaction and usability, ultimately driving up user engagement and sales. As mentioned in previous paragraphs, the integration of a complete MLOps workflow and the implementation to reach GDPR compliances are also interesting topics for follow-up research.

To sum up, this work succeeded in building a virtual try-on system known as GARMENTO. Throughout the development of the system, we gained valuable insights into the challenges and opportunities faced by industry stakeholders, which served as the foundation for the subsequent design and implementation phases of GARMENTO. We explored the use of the microservice architecture in building modular software and ended up with a comprehensive design which allows for nearly seamless integration into existing software ecosystems in fashion companies. We also succeeded in making a partial MLOps flow work with business services. We found that the proposed GARMENTO system has fulfilled the most fundamental functionalities of an enterprise virtual try-on system, therefore it can serve as the guidelines for companies to develop similar virtual try-on systems. However, implementation for specific businesses has to take into consideration the available software ecosystem to make the most appropriate decisions on the internal workflows to be supported by the GARMENTO system. For reference, the source code of the GARMENTO is publicly available on Github [[36]](https://www.zotero.org/google-docs/?0xWLui).

Appendix

**Appendix A**. System test cases

List of test cases used in the development of the GARMENTO system is available publicly at: <https://bit.ly/3W4mXUM>. Currently, all critical paths have been tested and passed all the test cases.

**Appendix B.** Latest source code directory

In addition to the monorepo provided in the References list, we provide an organization that clearly shows the decomposed and developed microservices. The Github organization is available at: <https://github.com/garmento-microservices>.

References

[[1] “Virtual Try-On for Glasses & Sunglasses | Online Try On | Eyeconic.” Accessed: Feb. 20, 2024. [Online]. Available: https://www.eyeconic.com/help-me/virtual-try-on](https://www.zotero.org/google-docs/?IeCaKM)

[[2] “Virtual Makeup Try-On - Virtual Makeover - Maybelline.” Accessed: Feb. 20, 2024. [Online]. Available: https://www.maybelline.com/virtual-makeover-makeup-tools](https://www.zotero.org/google-docs/?IeCaKM)

[[3] “Virtual Makeup Try-On - L’Oréal Paris.” Accessed: Feb. 20, 2024. [Online]. Available: https://www.lorealparisusa.com/virtual-try-on-makeup](https://www.zotero.org/google-docs/?IeCaKM)

[[4] “How CWS Elevated the B2B Workwear Shopping Experience with Virtual Try-On - Reactive Reality.” Accessed: Feb. 20, 2024. [Online]. Available: https://www.reactivereality.com/customers/b2b-workwear-shopping-experience-virtual-try-on](https://www.zotero.org/google-docs/?IeCaKM)

[[5] H. Sun, “Virtual Try-On Methods: A Comprehensive Research and Analysis,” presented at the 2023 International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2023), Atlantis Press, Nov. 2023, pp. 339–346. doi: 10.2991/978-94-6463-300-9\_35.](https://www.zotero.org/google-docs/?IeCaKM)

[[6] X. Han, Z. Wu, Z. Wu, R. Yu, and L. S. Davis, “VITON: An Image-Based Virtual Try-On Network”.](https://www.zotero.org/google-docs/?IeCaKM)

[[7] B. Wang, H. Zheng, X. Liang, Y. Chen, L. Lin, and M. Yang, “Toward Characteristic-Preserving Image-based Virtual Try-On Network.” arXiv, Sep. 12, 2018. Accessed: Feb. 21, 2024. [Online]. Available: http://arxiv.org/abs/1807.07688](https://www.zotero.org/google-docs/?IeCaKM)

[[8] M. R. Minar, T. T. Tuan, H. Ahn, P. L. Rosin, and Y.-K. Lai, “CP-VTON+: Clothing Shape and Texture Preserving Image-Based Virtual Try-On”.](https://www.zotero.org/google-docs/?IeCaKM)

[[9] H. Yang, R. Zhang, X. Guo, W. Liu, W. Zuo, and P. Luo, “Towards Photo-Realistic Virtual Try-On by Adaptively Generating$\leftrightarrow$Preserving Image Content.” arXiv, Mar. 12, 2020. Accessed: Feb. 21, 2024. [Online]. Available: http://arxiv.org/abs/2003.05863](https://www.zotero.org/google-docs/?IeCaKM)

[[10] M. R. Minar, T. T. Tuan, and H. Ahn, “CloTH-VTON+: Clothing Three-Dimensional Reconstruction for Hybrid Image-Based Virtual Try-ON,” *IEEE Access*, vol. 9, pp. 30960–30978, 2021, doi: 10.1109/ACCESS.2021.3059701.](https://www.zotero.org/google-docs/?IeCaKM)

[[11] A. Neuberger, E. Borenstein, B. Hilleli, E. Oks, and S. Alpert, “Image Based Virtual Try-On Network From Unpaired Data,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 5183–5192. doi: 10.1109/CVPR42600.2020.00523.](https://www.zotero.org/google-docs/?IeCaKM)

[[12] A. Cui, D. McKee, and S. Lazebnik, “Dressing in Order: Recurrent Person Image Generation for Pose Transfer, Virtual Try-On and Outfit Editing”.](https://www.zotero.org/google-docs/?IeCaKM)

[[13] K. Li, M. jin Chong, J. Zhang, and J. Liu, “Toward Accurate and Realistic Outfits Visualization with Attention to Details.” arXiv, Jun. 11, 2021. Accessed: Feb. 21, 2024. [Online]. Available: http://arxiv.org/abs/2106.06593](https://www.zotero.org/google-docs/?IeCaKM)

[[14] K. Li, J. Zhang, S.-Y. Chang, and D. Forsyth, “Wearing the Same Outfit in Different Ways -- A Controllable Virtual Try-on Method.” arXiv, Nov. 28, 2022. Accessed: Feb. 21, 2024. [Online]. Available: http://arxiv.org/abs/2211.16989](https://www.zotero.org/google-docs/?IeCaKM)

[[15] “PICTOFiT - #1 Virtual Try On Plugin for Fashion Online Shops.” Accessed: Feb. 21, 2024. [Online]. Available: https://pictofit.com/](https://www.zotero.org/google-docs/?IeCaKM)

[[16] R. Studd, “The Textile Design Process,” *Des. J.*, vol. 5, no. 1, pp. 35–49, Mar. 2002, doi: 10.2752/146069202790718567.](https://www.zotero.org/google-docs/?IeCaKM)

[[17] R. Roy and D. Wield, *Product Design and Technological Innovation: A Reader*. Open University Press, 1986.](https://www.zotero.org/google-docs/?IeCaKM)

[[18] “Building Microservices, 2nd Edition [Book].” Accessed: Feb. 26, 2024. [Online]. Available: https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/](https://www.zotero.org/google-docs/?IeCaKM)

[[19] C. Richardson, *Microservices patterns: with examples in Java*. Shelter Island, New York: Manning Publications, 2019.](https://www.zotero.org/google-docs/?IeCaKM)

[[20] S. Brown, “Modular Monoliths,” presented at the GOTO 2018, Berlin, Germany, Nov. 01, 2018.](https://www.zotero.org/google-docs/?IeCaKM)

[[21] “ONNX Concepts - ONNX 1.16.0 documentation.” Accessed: Feb. 22, 2024. [Online]. Available: https://onnx.ai/onnx/intro/concepts.html](https://www.zotero.org/google-docs/?IeCaKM)

[[22] “ONNX | Supported Tools.” Accessed: Feb. 22, 2024. [Online]. Available: https://onnx.ai/supported-tools.html#buildModel](https://www.zotero.org/google-docs/?IeCaKM)

[[23] “Why MLOps (and not just ML) Is Your Business’ New Competitive Frontier | AI Business.” Accessed: Feb. 22, 2024. [Online]. Available: https://aibusiness.com/verticals/why-mlops-and-not-just-ml-is-your-business-new-competitive-frontier](https://www.zotero.org/google-docs/?IeCaKM)

[[24] “Practical MLOps [Book].” Accessed: Feb. 13, 2024. [Online]. Available: https://www.oreilly.com/library/view/practical-mlops/9781098103002/](https://www.zotero.org/google-docs/?IeCaKM)

[[25] “MLflow: A Tool for Managing the Machine Learning Lifecycle — MLflow 2.10.2 documentation.” Accessed: Feb. 23, 2024. [Online]. Available: https://mlflow.org/docs/latest/index.html](https://www.zotero.org/google-docs/?IeCaKM)

[[26] “Kubeflow Documentation.” Accessed: Feb. 23, 2024. [Online]. Available: https://www.kubeflow.org/docs/](https://www.zotero.org/google-docs/?IeCaKM)

[[27] “Valohai | The MLOps platform for ML pioneers.” Accessed: Feb. 23, 2024. [Online]. Available: https://valohai.com/product/](https://www.zotero.org/google-docs/?IeCaKM)

[[28] “The C4 model for visualising software architecture.” Accessed: Feb. 26, 2024. [Online]. Available: https://c4model.com/](https://www.zotero.org/google-docs/?IeCaKM)

[[29] S. Brown, *Software Architecture for Developers*. Leanpub, 2012. Accessed: Feb. 27, 2024. [Online]. Available: https://leanpub.next/software-architecture-for-developers](https://www.zotero.org/google-docs/?IeCaKM)

[[30] “Docker Compose overview,” Docker Documentation. Accessed: Mar. 24, 2024. [Online]. Available: https://docs.docker.com/compose/](https://www.zotero.org/google-docs/?IeCaKM)

[[31] “Kubernetes.” Accessed: Mar. 24, 2024. [Online]. Available: https://kubernetes.io/](https://www.zotero.org/google-docs/?IeCaKM)

[[32] “Spring Cloud Netflix.” Accessed: Mar. 24, 2024. [Online]. Available: https://cloud.spring.io/spring-cloud-netflix/reference/html/](https://www.zotero.org/google-docs/?IeCaKM)

[[33] “What are microservices?,” microservices.io. Accessed: Feb. 17, 2021. [Online]. Available: http://microservices.io/index.html](https://www.zotero.org/google-docs/?IeCaKM)

[[34] “Microservices Pattern: Decompose by subdomain,” microservices.io. Accessed: Feb. 17, 2021. [Online]. Available: http://microservices.io/patterns/decomposition/decompose-by-subdomain.html](https://www.zotero.org/google-docs/?IeCaKM)

[[35] “Microservices Pattern: Remote Procedure Invocation (RPI),” microservices.io. Accessed: Mar. 24, 2024. [Online]. Available: http://microservices.io/patterns/communication-style/rpi.html](https://www.zotero.org/google-docs/?IeCaKM)

[[36] B. Do, “Garmento: A microservice-based garment try-on platform.” 2024. Accessed: Mar. 26, 2024. [Online]. Available: https://github.com/binhdoitsme/garmento](https://www.zotero.org/google-docs/?IeCaKM)