AI-Driven Virtual Try-On System in E-Commerce

TECHNICAL REPORT

SUBMITTED BY

Ahmad Raza

AG 2021-ag-7998

ADVISED BY

Dr Azam Zia

A TECHNICAL REPORT SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENT FOR THE DEGREE OF

BACHLOR OF SCIENCE

IN

INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

FACULTY OF SCIENCES

UNIVERSITY OF AGRICULTURE FAISALABAD

DECLARATION

I hereby declare that the contents of the report **Ai Driven virtual Try on System in E-Commerce** ”are project of my own research and no part has been copied from any published source (except the references). I further declare that this work has not been submitted for award of any other diploma/degree. The university may take action if the information provided is found false at any stage. In case of any default the scholar will be proceeded against as per UAF policy.

_____

Ahmad Raza

CERTIFICATE

To,

The Controller of Examinations,

University of Agriculture,

Faisalabad.

The supervisory committee certify that **[Ahmad Raza] [AG 2021-ag-7998]**has successfully completed his project in partial fulfillment of requirement for the degree of BS. Information Technology under our guidance and supervision.

_____

Dr.Azam Zia

Supervisor

_____

Dr. Muhammad Ahsan Latif

Chairman,

Department of Computer Science

ACKNOWLEDGEMENT

I thank all who in one way or another contributed in the completion of this report. First, I thank to ALLAH ALMIGHTY, most magnificent and most merciful, for all his blessings. Then I am so grateful to the Department of Computer Science for making it possible for me to study here. My special and heartily thanks to my supervisor, *[Dr.Azam Zia]* who encouraged and directed me. His/her challenges brought this work towards a completion. It is with his/her supervision that this work came into existence. For any faults I take full responsibility. I am also deeply thankful to my informants. I want to acknowledge and appreciate their help and transparency during my research. I am also so thankful to my fellow students whose challenges and productive critics have provided new ideas to the work. Furthermore, I also thank my family who encouraged me and prayed for me throughout the time of my research. May the Almighty God richly bless all of you

ABSTRACT

E-commerce has revolutionized the retail sector, particularly in the fashion industry. However, a major drawback persists—customers cannot physically try on clothing before making a purchase. This gap results in uncertainty about fit and appearance, leading to reduced confidence, increased returns, and customer dissatisfaction. To address this issue, this project presents an AI-driven Virtual Try-On (VTO) system designed for use in e-commerce platforms. The system leverages a combination of computer vision, deep learning, and augmented reality (AR) technologies to simulate the experience of trying on clothes virtually.

Users can upload a photo or use a live camera feed to visualize how a garment would appear on their body. Key AI techniques used in the system include Thin Plate Spline (TPS) transformation, Geometric Matching Networks (GMNs), and Generative Adversarial Networks (GANs) for image synthesis. The system achieves high accuracy and realism with performance metrics such as SSIM (0.895), LPIPS (0.053), and FID (11.74). The backend is built with Flask, and the frontend uses React.js for an interactive interface.

This virtual try-on system aims to enhance the online shopping experience by increasing user confidence and significantly reducing return rates. It also offers scalable deployment and supports integration with existing e-commerce platforms.

Table of Contents

List of Figures

List of Tables

Chapter 1 - INTRODUCTION

1.1 Background:

The rapid growth of e-commerce has reshaped the way consumers shop, offering unprecedented convenience and variety. In particular, the fashion retail sector has experienced significant transformation through online platforms. However, the lack of physical interaction with products remains a core limitation. Shoppers cannot assess how garments will look or fit on their bodies before purchase. This results in lower buyer confidence and contributes to high return rates, which in turn lead to increased operational costs and environmental waste.

The concept of a Virtual Try-On (VTO) system emerges as a promising solution to this problem. By incorporating technologies such as Artificial Intelligence (AI), Computer Vision, and Augmented Reality (AR), a VTO system simulates how clothes would appear on a user's body in real-time or from uploaded images. This AI-powered solution bridges the gap between online and physical shopping, enhances the user experience, reduces returns, and drives greater satisfaction. This project investigates and implements such a system tailored for the fashion e-commerce domain.

1.2 Description:

This project focuses on developing an AI-driven Virtual Try-On system that enables users to try clothes digitally using a photo or live camera. The main goal is to enhance the customer’s shopping experience by allowing them to visualize how a particular outfit would look on their own body before making a purchase. The system achieves this by combining pose detection, image warping, and clothing alignment into a seamless pipeline.

The backend uses deep learning models such as GANs and GMNs to perform clothing synthesis and image transformation. Pose estimation is done using OpenPose, while Thin Plate Spline (TPS) warping is used to align clothing on the user's body. The frontend allows users to interact with the system via a simple React.js-based interface, where they can upload images, select garments, and receive try-on results.

1.3 Problem Statement:

.The fashion e-commerce industry struggles with a major shortcoming: users cannot physically try on clothing before purchase. As a result, customers often face uncertainty about how the garments will fit or appear on their unique body shapes. This limitation leads to reduced confidence, increased product returns, and ultimately, lower customer satisfaction. Traditional images and sizing charts fail to provide a personalized shopping experience. A solution is needed to virtually simulate how clothing would look on an individual in real-time or via uploaded images, enhancing interactivity and decision-making for online shoppers.

1.4 Scope:

This project is limited to implementing a web-based AI-powered virtual try-on solution focused on the fashion domain. Users can upload an image or use a camera to see how selected garments appear on their body. The system supports integration with basic e-commerce functionalities like browsing, adding to cart, and checkout. However, the current scope does not extend to 3D modeling, dynamic fabric movement, or mobile app deployment. Future improvements may cover these aspects, along with real-time webcam try-on features. The project is primarily aimed at desktop/web usage and assumes access to a stable internet connection and a device capable of running AI inference.

1.5 Objectives:

To develop an AI-driven virtual try-on system that allows users to visualize clothing on their own image, enhancing the online shopping experience.

- To build a web-based interface for virtual try-on using uploaded user images.
- To implement deep learning models (GANs, TPS, GMNs) for realistic garment fitting.
- To extract user body pose using pose estimation tools like OpenPose.
- To align and overlay clothing images onto user photos with high accuracy.
- To reduce customer return rates and increase buyer confidence through visual feedback.
- To integrate the virtual try-on system with a basic e-commerce backend.

1.6 Feasibility:

1.6.1 Technical Feasibility – The technologies required, such as deep learning frameworks (PyTorch), pose estimation (OpenPose), and image warping (TPS), are readily available and compatible with the system. The model has been successfully trained and tested using GPUs like NVIDIA RTX 3090, and integration with web platforms (React.js + Node.js) is technically achievable.

1.6.2 Schedule Feasibility - The project was completed over the span of two semesters, with time allocated for system design, model development, training, testing, and deployment. Gantt charts were used to ensure timely task completion.

1.6.3 Economic Feasibility – Development was done using free and open-source tools (Google Colab, PyTorch, etc.), making it cost-effective. Only a capable GPU system was required for training, which was accessed using Google Colab Pro or university resources.

1.6.4 Cultural Feasibility – The system is designed to work for diverse users by enabling personalized try-ons. There is potential to support culturally specific garments in future, making it adaptable to various cultural preferences.

1.6.5 Legal/Ethical Feasibility – User images are only processed temporarily and can be deleted post-usage to maintain privacy. Consent mechanisms and disclaimers are included. Legal risks around data storage and image processing are mitigated using secure APIs and local execution (no public cloud usage)..

1.6.6 Resource Feasibility – Required resources include a development machine with GPU, basic frontend and backend tech stack (React, Node.js), and access to training datasets like VITON-HD. All were available during the project.

1.6.7 Operational Feasibility – The final system works in real-world conditions with live user uploads. It successfully generates try-on previews with acceptable accuracy and performance, proving its practical usability for deployment in an e-commerce website.

1.7 Requirements:

1.7.1 Functional Requirements

The following functional requirements define the core operations of the AI-driven virtual try-on system. These requirements describe the expected behavior of the system under various conditions:

FR01: Provide user name and password to log in

FR01-01	System shall provide a signup/login form.
FR01-02	System shall authenticate credentials and allow access.
FR01-03	Invalid login attempts shall display an error.

FR02: Create user account

FR02-01	System shall allow admin to add products (images, details).
FR02-02	System shall allow users to browse product categories.

FR03: Virtual Try-On

FR01-01	User shall upload a photo..
FR01-02	System shall apply the selected garment on the user image.
FR01-03	System shall display the try-on output image.

1.7.2 Non- Functional Requirements

Non-functional requirements define how the system behaves and outline the quality attributes it must meet. These include performance, usability, reliability, availability, and security constraints that ensure a smooth and effective user experience. The non-functional requirements for the AI-driven virtual try-on system are as follows:

NFR01: System shall be responsive and support all major web browsers.
NFR02: System shall process try-on results in less than 5 seconds (for 512x512 images).
NFR03: System shall secure user data using HTTPS and authentication.
NFR04: System shall be available 24/7 on the deployed server.
NFR05: System shall support image input formats: .jpg, .png.

1.7.3 Hardware Requirements

Processor: Intel Core i7 or equivalent
RAM: Minimum 8GB (32GB for model training)
GPU: NVIDIA RTX 3060 or higher (for training)
Storage: 100GB free space
Internet: Required for image upload and inference API

Tool	Similar Tools	Why Selected/Not Selected
NVIDIA RTX 3090	NVIDIA RTX 4080, A100	RTX 3090 offers a balance of cost and performance for high-resolution image synthesis.
Intel i7-11700K	AMD Ryzen 7 5800X	Chosen for its compatibility with development tools and balanced single-thread and multi-thread performance.
Google Colab	Kaggle Kernels, AWS EC2	Free GPU access for experimentation, but limited runtime compared to AWS EC2.

Table 1.7.3: Hardware Tools

1.7.4 Software Requirements

OS: Windows 10 / Ubuntu 20.04
Browser: Google Chrome, Mozilla Firefox
Frameworks: React.js (Frontend), Node.js (Backend), PyTorch (AI Model)
Libraries: OpenCV, OpenPose, NumPy, Flask API

Tool	Similar Tools	Why Selected/Not Selected
PyTorch	TensorFlow	PyTorch offers easier debugging, dynamic computation graphs, and better community support for GANs.
React.js	Angular, Vue.js	React.js provides a lightweight, component-based structure ideal for fast prototyping.
MongoDB	MySQL, PostgreSQL	Chosen for its NoSQL flexibility to handle unstructured data (e.g., user photos, product categories).
OpenPose/DensePose	Mediapipe, BlazePose	OpenPose/DensePose provides state-of-the-art pose estimation accuracy critical for this application.
Node.js	Django, Flask	Node.js works seamlessly with MongoDB and offers better scalability for backend services.

Table 1.7.4: Software Tools

1.8 Stakeholders:

Users/Customers: End-users who interact with the system to try on clothing.
Admin: Manages the product catalog and user database.
Developers: Responsible for system development, training, and deployment.
E-commerce Business Owners: Potential adopters of the technology for integration.
Researchers: Interested in virtual try-on advancements and AI-driven interaction.

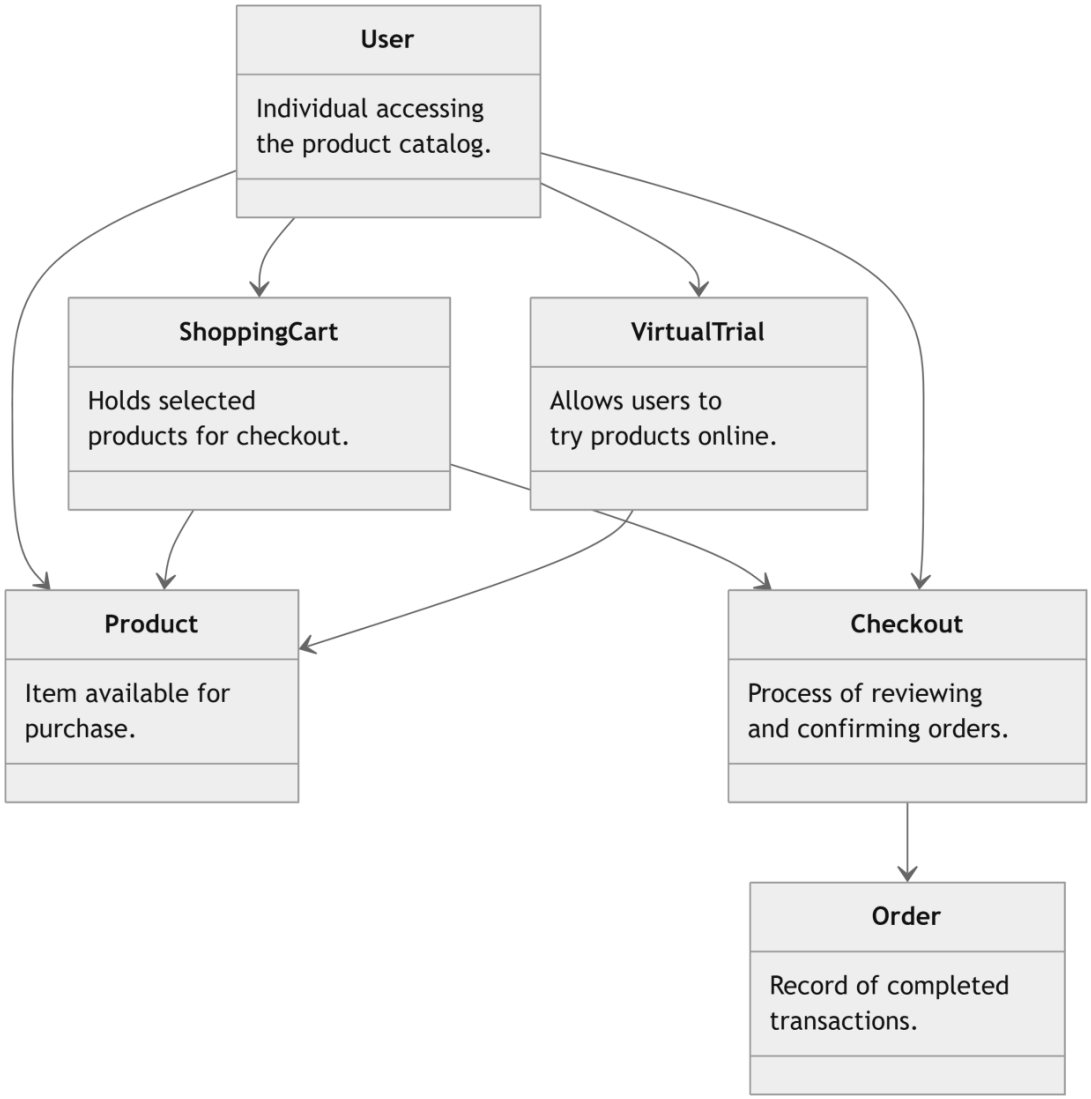


Figure 01 Stakeholders

Chapter 2 - METHODOLOGY

2.1 Process Model:

For the development of the **AI-Driven Virtual Try-On System**, we adopted the **Agile Software Development Model**, specifically a **Scrum-based approach**, due to its flexibility, iterative structure, and emphasis on customer feedback. The nature of our project demanded continuous testing and refinement of features such as image synthesis, clothing alignment, and frontend interaction — all of which benefitted greatly from Agile's incremental delivery.

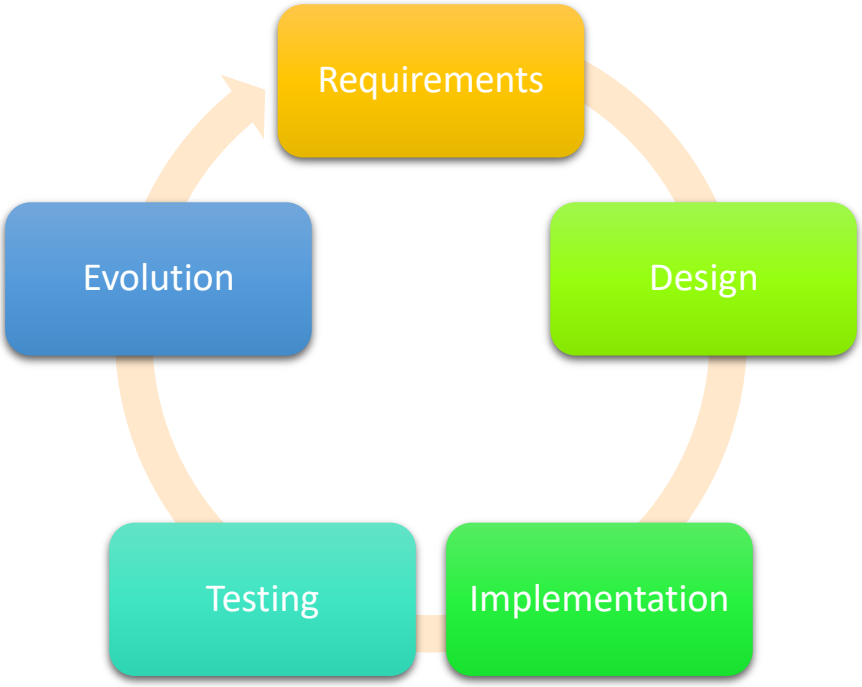


Figure 0.1 Agile Activities

2.2 Tools & Technologies

Our project leveraged a combination of **AI frameworks**, **web development tools**, and **cloud resources** to build an end-to-end try-on solution. Below is a categorized overview:

Tool	Reason for Selection
NVIDIA RTX 3090	Provided 24GB VRAM for high-res image processing and GAN training
Intel i7 11700K	Balanced single/multi-thread performance for backend & local dev
32GB DDR4 RAM	Supported fast model inference and data preprocessing

Tool	Reason for Selection
1TB SSD	Ensured high-speed file I/O during training and testing
Google Colab	Used for initial model experimentation with free GPU access

2.3 Design:

Note: Given below are some generic software design diagrams. However more design/UML content can be added according to the requirement and nature of your project.

2.3.1 Use Case Diagrams:

In the AI-Driven Virtual Try-On System, multiple interactions between users, the system, and admin roles were identified. These use cases revolve around uploading user images, selecting clothing items, generating try-on previews, managing products, and completing purchases.

The components in a use case diagram include:

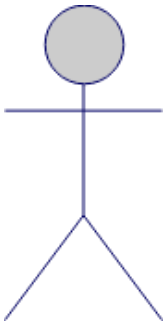
Actors:

Actors are first thing you need to find for the use case diagram. Actors represent external entities of the system. These can be people or things, such as external hardware that interact with the system. For example, if an online store is being modeled there can be more than one actor that interacts with the store functionality. Such as the Customer and stocker will be the actors in the system. It is represented simply by a stick figure with its name at the bottom of it.

User: Uploads image, selects clothes, views try-on result, places orders.

Admin: Manages products, handles backend configurations.

System: Processes try-on requests, performs pose estimation, synthesizes images.



Actor

Use Cases:

Use cases are functional parts of the system. They figure out what actions/functionalities a user will perform. Use cases are basically the functional requirements that you have pointed out in the functional and non-functional requirements topic. For example: The customer "browses the catalog", "chooses items to buy", and "pays for the items". Here browse catalog, buy item and pay for item are the use cases. Many actors can share a single use cases. The notation for a use case is an ellipse. As it is displayed below:

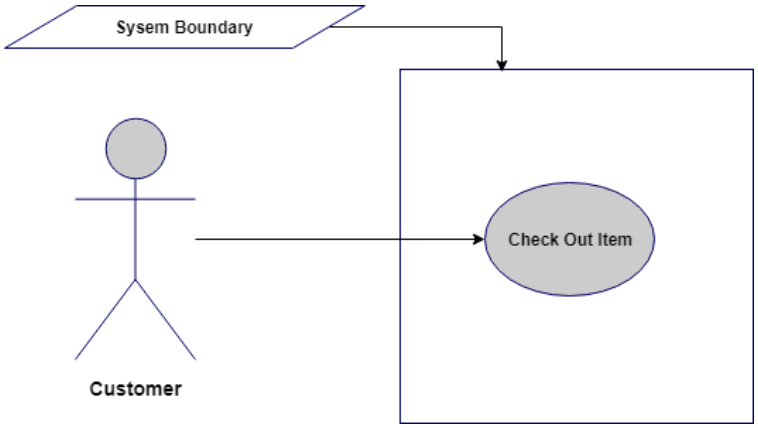
- Upload Personal Image
- Select Clothing Item
- View Try-On Result
- Add to Cart
- Checkout Order
- Manage Products (Admin)
- Authenticate User (Login/Signup)

Associations:

Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever there is direct interaction between actor and use case. Associations are modeled as lines connecting use cases and actors to one another, with an arrowhead on one end of the line.

System boundary:

The system boundary encapsulates the entire try-on and shopping workflow, with external actors (users/admin) interacting through web UI or API.



Relationship between Use cases:

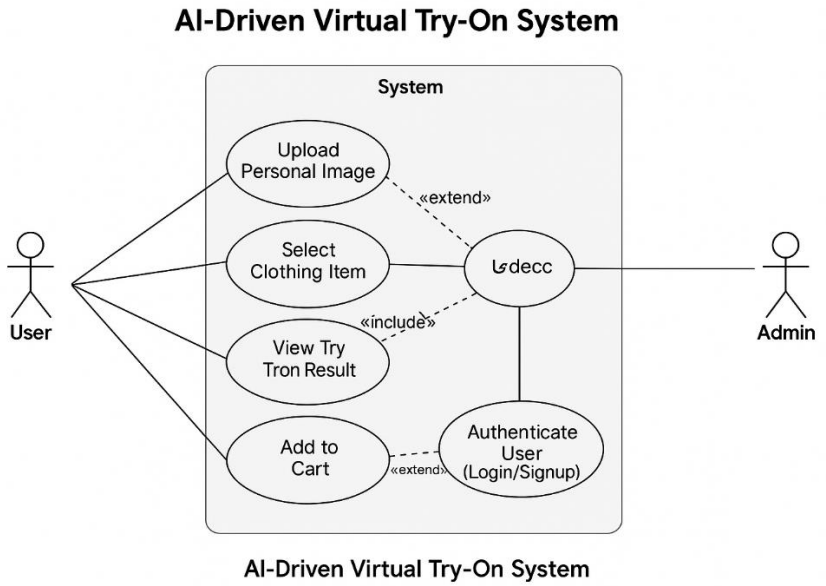
i) Include/Uses:

Include relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion use case). <<include>> use cases must be **used** by the use cases that **use** them before the latter can be complete Used where one use case always invokes another (e.g., "Checkout Order" includes "Add to Cart")

ii) Extend:

A use case **extends** another use case to do more than the latter. It extends the functionality of one use case to further level. is displayed in the diagram editor as a dashed line with an open arrowhead pointing from the extension use case to the base use case. The arrow is labeled with the keyword «extend».

Optional or conditional flows (e.g., "View Try-On Result" may extend to "Download Image")



Use Case Diagram:

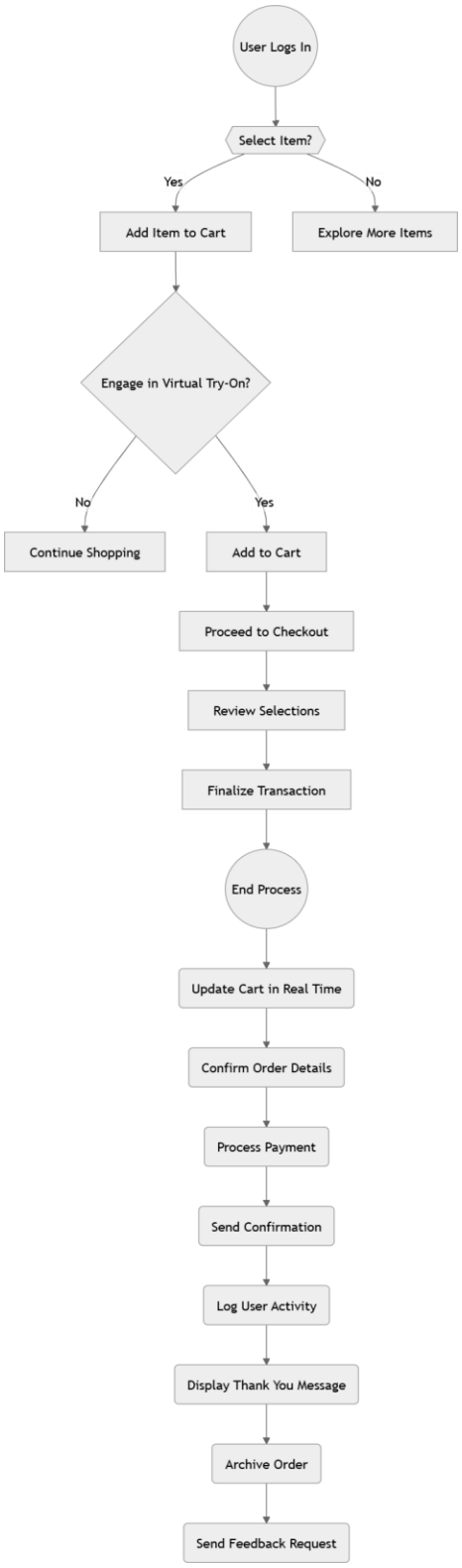


Figure 2.02 Use Case Diagram

2.3.2 Usage Scenario

Usage scenario is the actual text-based representation of the use case, among various representation methods discussed above. A usage scenario is likely to have various sections depending upon the level of details required in a given system. There is no fixed standard so far for number of sections in a use case (usage scenario).
Following is a typical table structure for usage scenario, note that it is not mandatory to write a usage scenario in the table format only, and it is likely that you will find different structures for the same representation.

Use Case Title	**Write Title Here (must match use case title in use case diagram)**
Abbreviated Title	Upload Personal Image
Use Case Id	UC-01
Requirement Id	FR01
Description: This use case allows a user to upload their personal image (front-facing) to be used for generating the virtual try-on preview.	
PreConditions:- User must be logged into the system. Valid image file (JPG, PNG) must be available.	
Task Sequence	**Exceptions**
1. User navigates to the try-on page.	Invalid file format.
2. System prompts user to upload an image.	Image upload fails due to network error.
3. User selects and uploads an image file	File size exceeds limit.
4. System validates and stores the image.	
.	
Post Conditions: The image is saved and linked to the user profile.	
Unresolved issues: None	
Authority: User	
Modification history: 1.0 **Author:** Ahmad Raza (2021-ag-7998) **Description:** Usage scenario for the image upload module in the virtual try-on workflow.	

Let's explore each section from the template provided above. (you have to make only usage scenario tables, below is description of its each part)

- **Use Case Title**
  It is the name or label of the use case for which we are writing the usage scenario. Generally it must start with a Verb and it should consist of 2 to 4 words e.g. *Add User*, *Manage User Roles* etc.

Use Case Title	Select Clothing Item

- **Use Case Id**

  Sometimes use cases are indexed for better reference in overall project documentation/artifacts. This can be any form of series e.g. 1, 2, 3 etc. Priority based use case id is another famous use for this section. Use cases are indexed to present their importance in the system. You would want to set ascending or descending rating or priority for all use cases i.e. the most important use cases are ranked higher so that the project team knows what should be implemented first in the upcoming phases/deliverable of the project.
- **Requirement Id**

  The purpose of this section is same as 'Use Case Id' section. The number written against this section represents the corresponding functional requirements this use case belongs to. It is compulsory to index all the functional requirements properly before this section can be used:

Requirement Id	FR02

- **Description**

  It should be a very brief description of the use case under discussion. Generally this portion should consist of 2/3 lines.

Description	This use case enables the user to browse available clothing items from the product catalog and select one for virtual try-on.

- **Pre-Conditions**

  This section should enlist what must be true before this use case can be performed.

Pre Conditions: 1. User must be logged in.. 2. Clothing items must already be added to the system database by admin.

- **Task Sequence & Exceptions**

  This is the most important section of the usage scenario. It is also referred as Success Scenario, Actions, (or simply) Scenario. This should be a list of actor's interaction with the use case.

Task Sequence	**Exceptions**
1. User opens the clothing catalog.	No Products.
2. System displays product categories.	Available in the
3. User selects a category and views product items.	Selected category
4. System after confirmation adds the new account.	Database query
5. User selects a specific clothing item.	Failure
6. A log is saved on the successful operation of the use case.	

Alternate scenarios could be more than one; in this case, it will be better to make a bold heading to show each alternate scenario separately. Again, there can be multiple ways to show the alternate scenarios.

Exceptions are any unhandled scenarios that must be discussed under this section. Sometimes there are ambiguous situations in start of project that might hurdle the flow of events in the Task Sequence portion. In such situations the details are provided in the Exceptions section. Generally, this section should be left blank as in case of final project, you will get fixed requirements at the start and thus there should be no ambiguity.

- **Post Conditions**

  The conditions that must be true depending upon the successful use case are mentioned in this section.

Post Conditions:
– Selected clothing item is stored temporarily for try-on session.

- **Unresolved Issues**

  In addition to the Exceptions portion, we write unresolved issues (None) in this section, so that in later phases (when the situation gets more clear).

  Just like Exceptions section, this will be generally left blank (or its row can be deleted from the use case table-structure.
- **Authority**

  The role that is allowed to perform this use case, in our current example it will be Administrator.
- **Modification History**

  If a use case is updated in later stages of the project development, the versioning information should be mentioned in this section (version can be a series such as 1.0, 2.0, 3.0 or 1.1, 1.2, 1.3 etc.)
- **Author**

  Ahmad Raza (2021-ag-7998).

- **Description**

Describes how users interact with the system to select a clothing item for the virtual try-on

Use Case Title	View Try-On Result	
Use Case Id	UC-03	
Requirement Id	FR03	
Description: This use case enables the user to browse available clothing items from the product catalog and select one for virtual try-on.		
Pre Conditions: 1. User must be logged in.. 2. Clothing items must already be added to the system database by admin..		
Task Sequence		**Exceptions**
1. User opens the clothing catalog..		No products available in the selected category.
2. System displays product categories.		
3. User selects a category and views product items.		Database query failure.
User selects a specific clothing item.		
4. System after confirmation adds the new account.		
5. System sends the account creation email to the administrator's email id and user's email address.		
6. A log is saved on the successful operation of the use case.		
Post Conditions: – Selected clothing item is stored temporarily for try-on session.		
Unresolved issues: None		
Authority: User		
Modification history: 1.0 **Author:** Ahmad Raza (2021-ag-7998) **Description:** Covers the final stage in the virtual try-on flow where the AI-generated result is delivered to the user interface.		

Important Points:

- A **separate usage scenario** has been created for each main use case derived from the Use Case Diagram (e.g., Upload Image, Select Clothing, View Try-On, Add to Cart, Checkout).
- The **titles of the use cases** used in both the diagram and the usage scenarios are **kept consistent** to maintain traceability and clarity.
- Every usage scenario follows a **uniform format**, including critical sections such as **Pre-Conditions, Post Conditions, Task Sequence, Exceptions, and Authority**.
- **Task Sequences** were written from the user's and system's perspective, **avoiding low-level GUI actions** like "clicking buttons" or "displaying popups" unless they were functionally relevant.
- **Exceptions** and **Unresolved Issues** have been clearly separated to indicate what is already handled and what may still require further design clarification or future enhancement.

- **Use Case IDs (UC-01, UC-02...)** and **Requirement IDs (FR01, FR02...)** were systematically assigned to reflect **priority** and **function mapping**, ensuring better organization and easier reference.
- **Author and Modification History** are included in each table for **version tracking and authorship attribution**, aligned with professional documentation standards.

2.3.3 Sequence Diagram:

This sequence diagram illustrates the interaction between the user, frontend, backend server, AI module, and the database during the virtual try-on process.

Sequence Diagram Notations:

Actors – An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

User uploads an image and selects a clothing item.

Frontend (React.js) sends image and product ID to the backend (POST /tryon).

Backend Server (Node.js/Express) receives the request and calls the **AI Module**.

AI Module (PyTorch/GAN):

- Extracts pose using OpenPose.
- Warps clothing using TPS.
- Generates a try-on image using GAN.

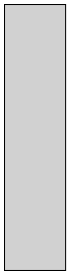
Backend stores generated image reference temporarily (optional).

Backend responds with the **generated try-on image**.

Frontend displays the try-on result to the user.

Class Roles or Participants:
Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

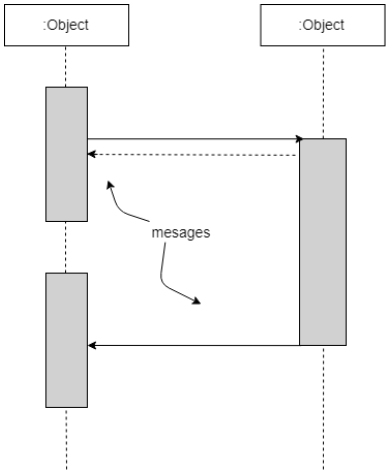
Activation or Execution Occurrence:
Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.



Activation or Execution Occurence

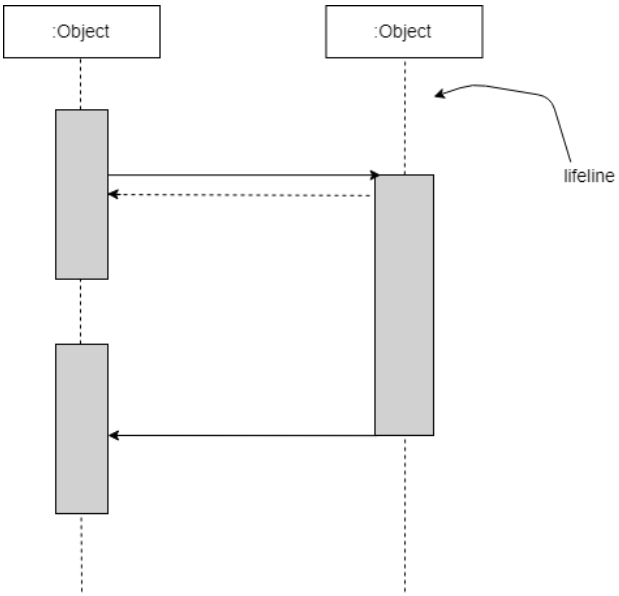
Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.



Lifelines

Represent how long each participant is active in the process..



Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].

Sequence Diagram Example (for item return use case):

Note: Make sequence diagram for each use case illustrated in use case diagram

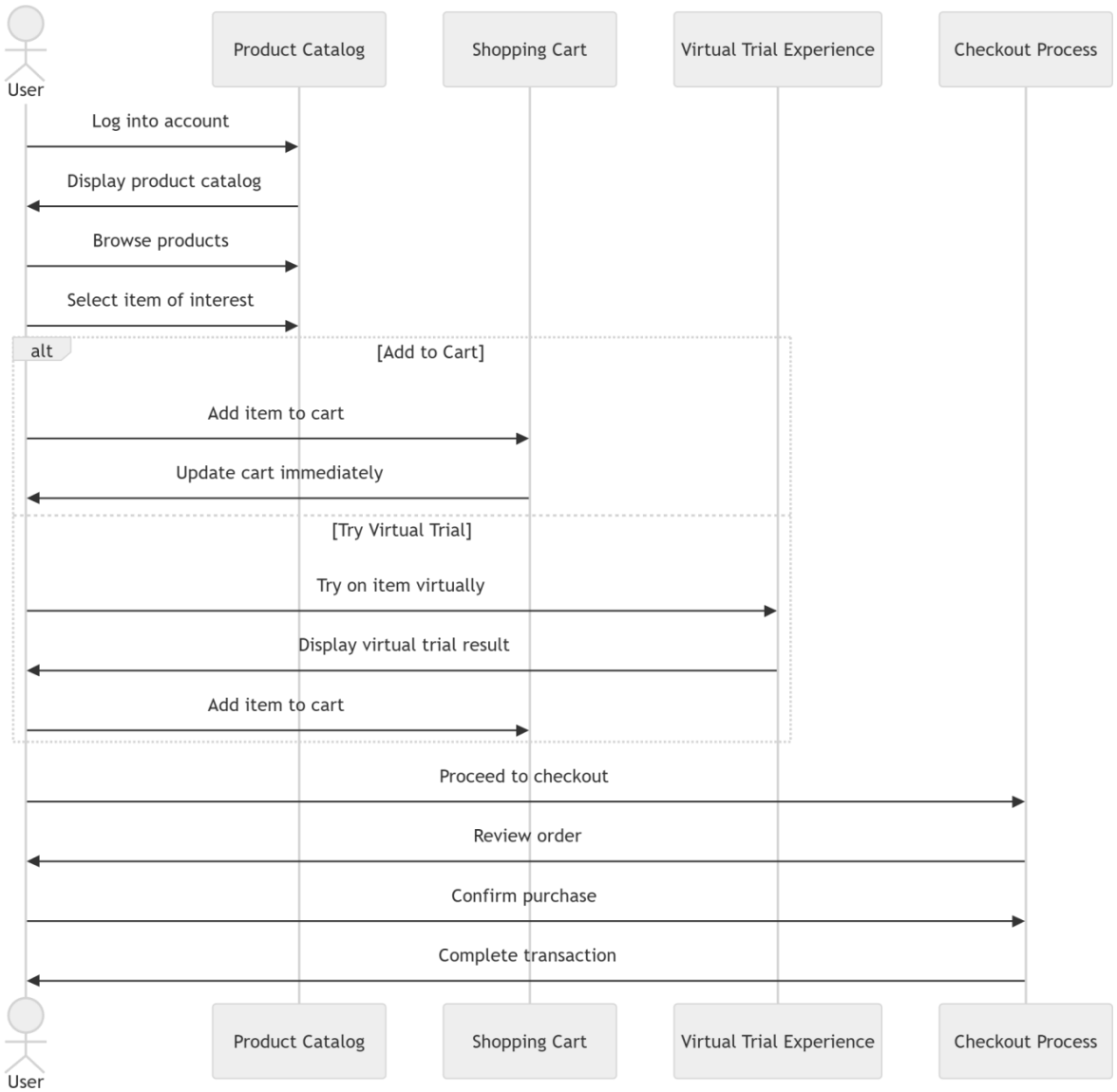


Figure 2.3 Sequence Diagram

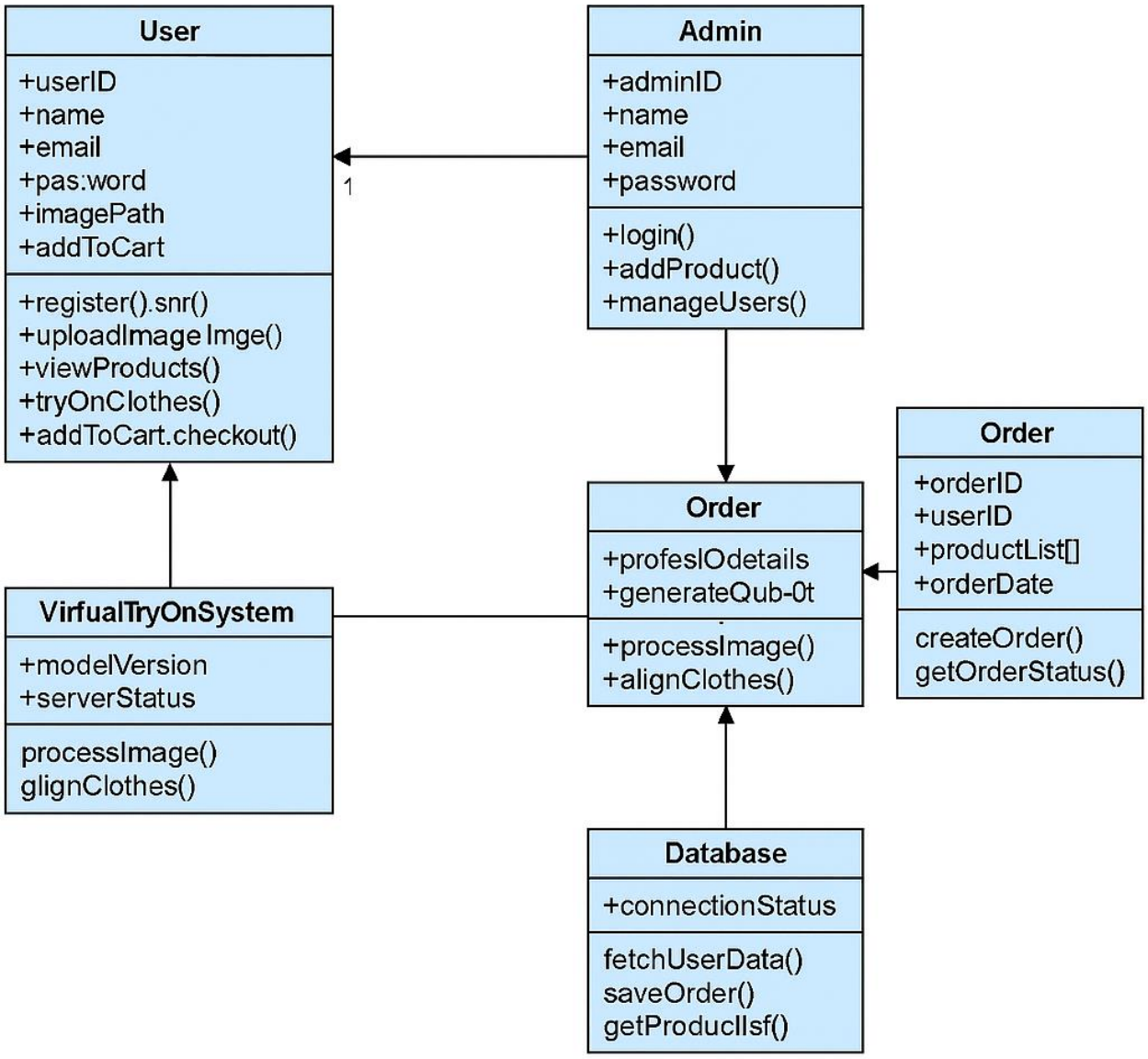
2.3.4 Class Diagram:

The class diagram below represents the static structure of the **AI-Driven Virtual Try-On System**. It outlines the main system classes, their attributes, methods, and relationships.

Classes & Descriptions:

1. **User**
   Attributes: userID, name, email, password, imagePath
   Methods: register(), login(), uploadImage(), viewProducts(), tryOnClothes(), addToCart(), checkout()
2. **Admin**
   Attributes: adminID, name, email, password
   Methods: login(), addProduct(), manageUsers()
3. **Product**
   Attributes: productID, name, category, price, size, imagePath
   Methods: getProductDetails(), updateStock()
4. **Order**
   - *Attributes*: orderID, userID, productList[], orderDate, status
   - *Methods*: createOrder(), getOrderStatus()
5. **VirtualTryOnSystem**
   - *Attributes*: modelVersion, serverStatus
   - *Methods*: processImage(), alignClothes(), generateOutput()
6. **Database**
   - *Attributes*: connectionStatus
   - *Methods*: fetchUserData(), saveOrder(), getProductList()

.



Class Diagram in vlass Diagram

2.3.5 Data Flow Diagram:

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. A Data Flow Diagram (DFD) is traditional visual representation of the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or combination of both.

This section describes how **data flows** between the components of your virtual try-on system using **DFD Level 0** (context level) and **DFD Level 1** (detailed system-level).

It is usually beginning with a context diagram as the level 0 of DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required. Progression to level 3, 4 and so on is possible but anything beyond level 3 is not very common. Please bear in mind that the level of details for decomposing particular function really depending on the complexity that function. For further reading use the link given below:
https://www.visual-paradigm.com/guide/data-flow-diagram/what-is-data-flow-diagram/

DFD Diagram Notations

External Entity

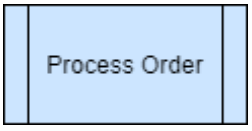
An external entity can represent a human, system or subsystem. It is where certain data comes from or goes to. It is external to the system we study, in terms of the business process. For this reason, people used to draw external entities on the edge of a diagram.

- **User:** Provides image input and receives try-on result.
- **Admin:** Manages products and views orders.
- **Payment Gateway (optional):** Processes payments during checkout.

Process

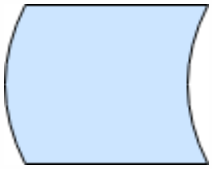
A process is a business activity or function where the manipulation and transformation of data takes place. A process can be decomposed to finer level of details, for representing how data is being processed within the process.

Virtual Try-On System: Central system that handles all requests.



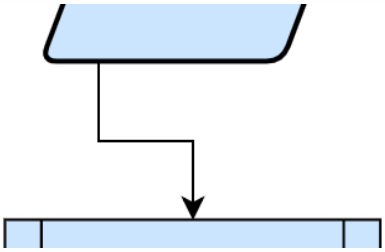
Data Store
A data store represents the storage of persistent data required and/or produced by the process. Here are some examples of data stores: membership forms, database table, etc.

- User DB
- Product DB
- Order DB
- Try-On Results



Data Flow
A data flow represents the flow of information, with its direction represented by an arrow head that shows at the end(s) of flow connector.

- User → Upload Image → System
- User → Select Product → System
- System → AI Engine → Try-On Result → User
- Admin → Add Product → System



Context Diagram:

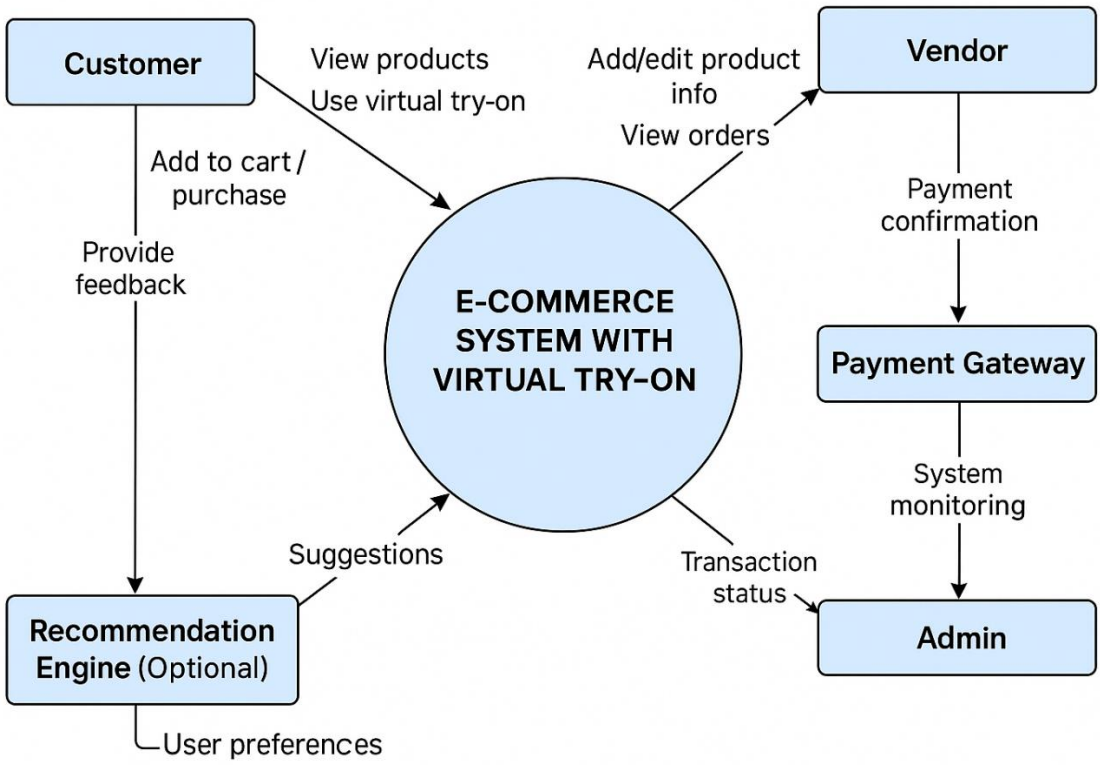


Figure 0.4 Context Diagram

Level 0:

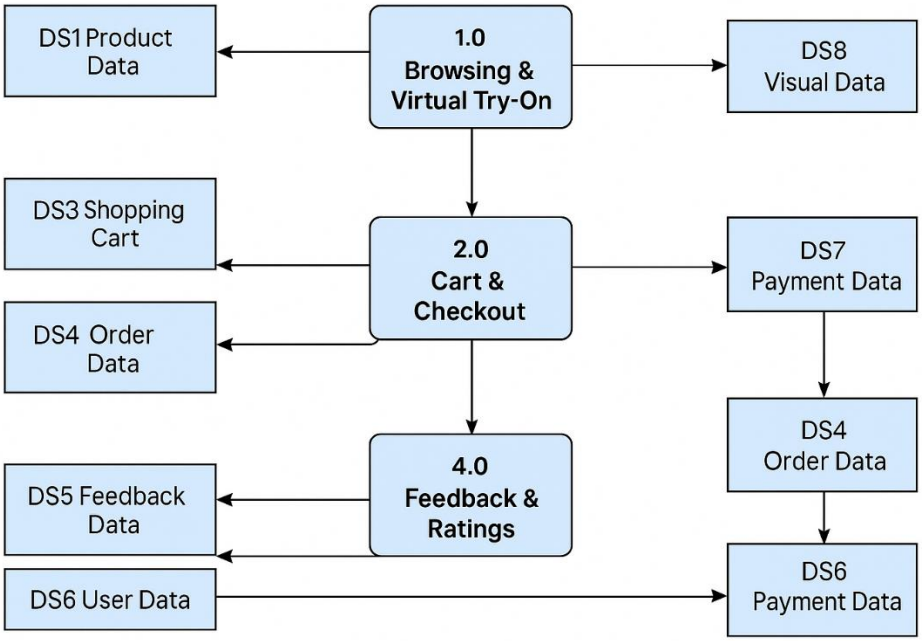


Figure 02.5 Level 0 DFD

Level 1:

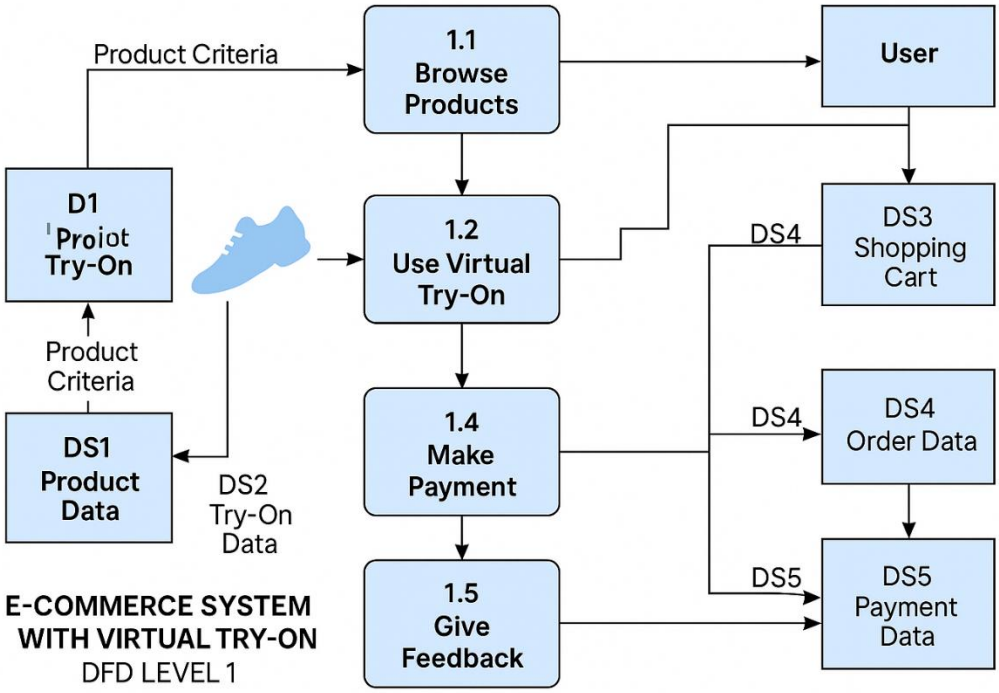


Figure 2.6 Level 1 DFD

2.3.6 ER Diagram:

This section describes the **data model** of your AI-Driven Virtual Try-On System using an **Entity Relationship Diagram (ERD)**. It outlines how entities like users, products, orders, and try-on sessions relate to each other.

1. **User.**
   - `user_id` (PK)
   - `name`
   - `email`
   - `password_hash`
   - `profile_image`

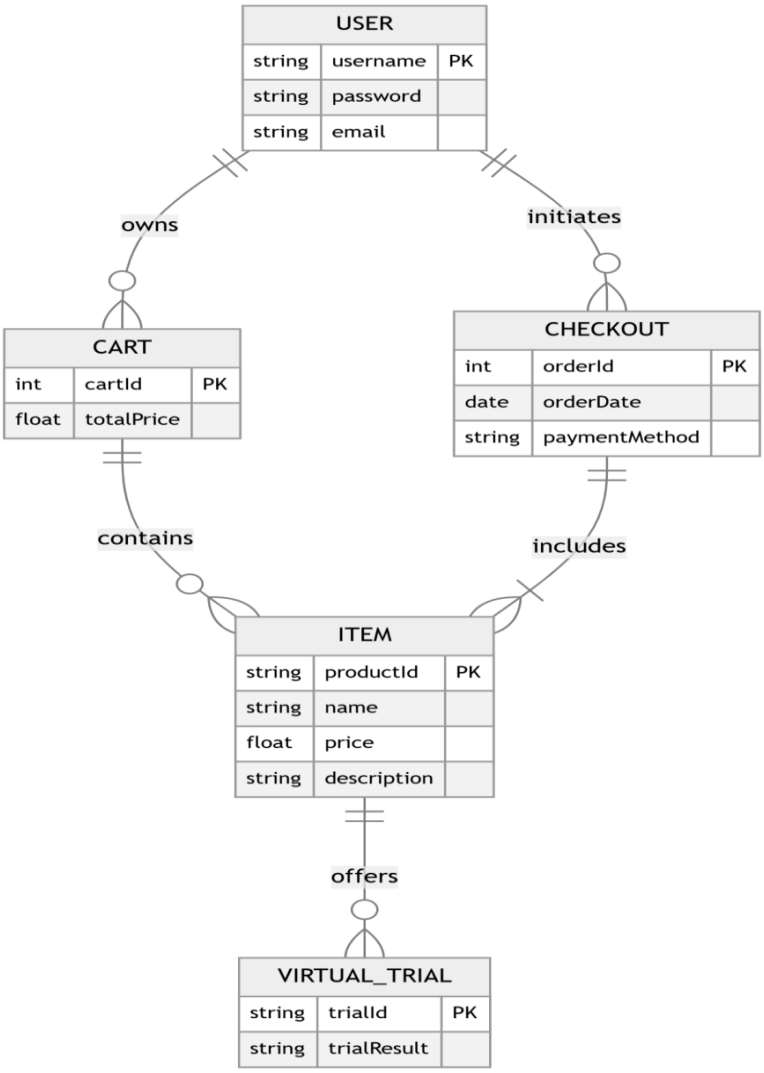


Figure 0.7 Entity Relationship Diagram

2.3.7 Database Model:

A database model shows the logical structure of a database, including the relationships and constraints that determine how data can be stored and accessed. Individual database models are designed based on the rules and concepts of whichever broader data model the designers adopt. Most data models can be represented by an accompanying database diagram. Below is an example for library management system. [4] To read more about designing database model visit: https://www.datanamic.com/support/lt-dez005-introduction-db-modeling.html

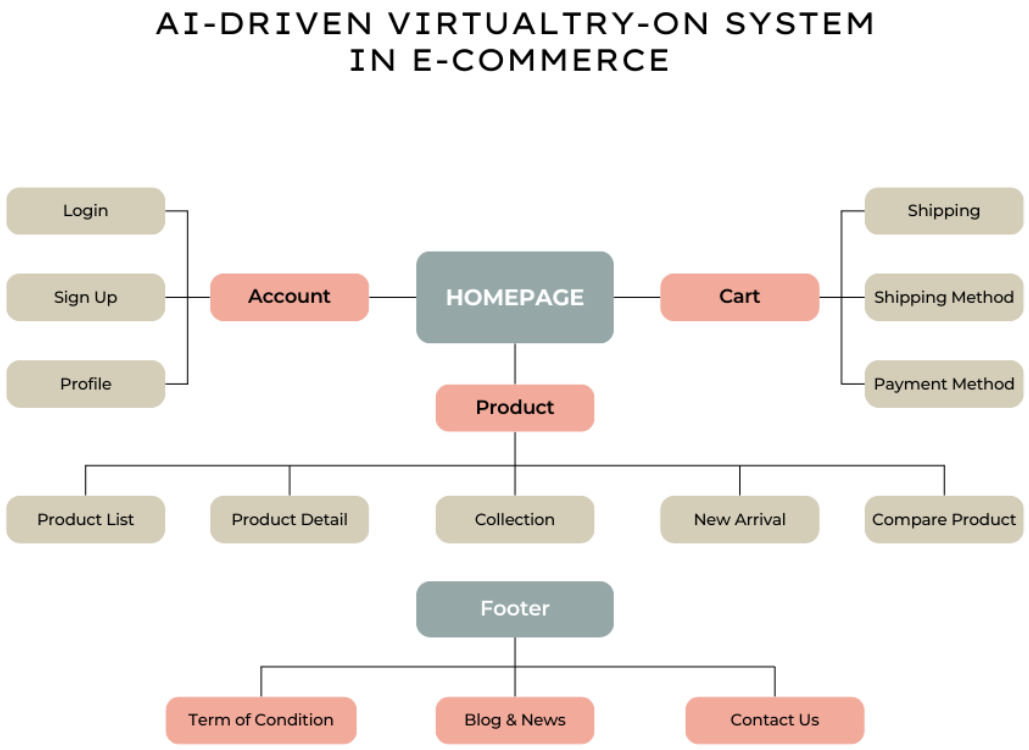


Figure 2.8 Database Model

2.3.8 Architecture:

3-Tier: (or N- Tier/multitier, chose whatever best suits your project nature)

A 3-tier architecture is a type of software architecture which is composed of three "tiers" or "layers" of logical computing. They are often used in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Doing so gives greater flexibility to development teams by allowing them to update a specific part of an application independently of the other parts. This added flexibility can improve overall time-to-market and decrease development cycle times by giving development teams the ability to replace or upgrade independent tiers without affecting the other parts of the system.

Presentation Tier- The presentation tier is the front-end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS, or through other popular web development frameworks, and communicates with other layers through API calls.

Application Tier- The application tier contains the functional business logic which drives an application's core capabilities. It's often written in Java, .NET, C#, Python, C++, etc.

Data Tier- The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

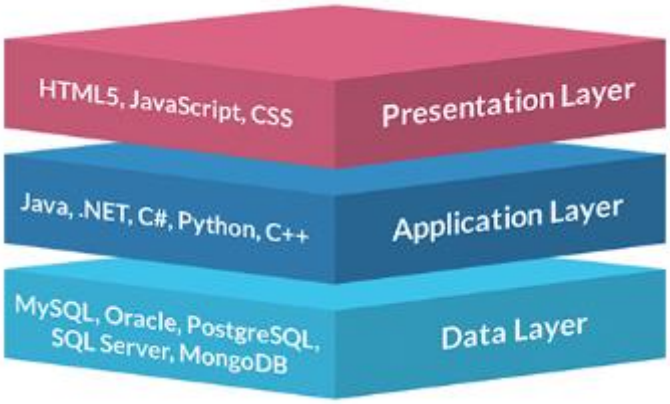


Figure 02.9 Application Architecture

Chapter 3 - RESULTS & DISCUSSION

This chapter presents a detailed evaluation of the AI-Driven Virtual Try-On System based on software testing methodologies, functional verification, and system-level observations. It verifies whether the developed application meets the functional and non-functional requirements defined in Chapter 1. The primary focus is to assess the correctness, usability, performance, and reliability of the application by executing well-defined test cases. Testing plays a vital role in ensuring the quality and success of the final product.

3.1 Testing:

Software testing is a systematic process that helps verify whether the implemented software fulfills its intended goals and behaves as expected under various conditions. In this project, testing was conducted to evaluate the system's ability to perform virtual try-ons, manage users, handle product selection, support image generation, and maintain seamless integration between backend and frontend.

The testing strategy involved a combination of **manual black-box testing** and functional verification through real user interaction scenarios. Testing was carried out by executing test cases derived from core functional requirements such as user login, image upload, clothing selection, virtual try-on result generation, cart management, and order placement. Each test case focused on verifying a specific action or flow within the application.

3.2 Test Cases:

Test cases need to be simple and transparent

Create test cases that are as simple as possible. They must be clear and concise as the author of the test case may not execute them.

Use assertive language like go to the home page, enter data, click on this and so on. This makes the understanding the test steps easy and tests execution faster.

Create Test Case with End User in Mind

The ultimate goal of any software project is to create test cases that meet customer requirements and is easy to use and operate. A tester must create test cases keeping in mind the end user perspective

Avoid test case repetition.

Do not repeat test cases. If a test case is needed for executing some other test case, call the test case by its test case id in the pre-condition column

Do not Assume

Do not assume functionality and features of your software application while preparing test case. Stick to the User Requirement Specification Documents.

Ensure 100% Coverage

Make sure you write test cases to check all software requirements mentioned in the specification document.

Test Cases must be identifiable.

Name the test case id such that they are identified easily while tracking defects or identifying a software requirement at a later stage. [2]

Test Case Template Help:

- ✓ Test case ID: Unique ID is required for each test case.
- ✓ Test Title/Name: Test case title. E.g. verify login page with a valid username and password.
- ✓ Test priority (Low/Medium/High): This is very useful while test execution. Test priority for business rules and functional test cases can be medium or higher whereas minor user interface cases can be of a low priority. Test priority should always be set by the reviewer.
- ✓ Requirements: Requirements for which this test case is being written for. Preferably the exact section number of the requirement doc.
- ✓ Test Summary/Description: Describe the test objective in brief.
- ✓ Test Execution Date: Date when the test was executed.
- ✓ Pre-conditions: Any prerequisite that must be fulfilled before the execution of this test case. List all the pre-conditions in order to execute this test case successfully.
- ✓ Dependencies: Mention any dependencies on the other test cases or test requirement.
- ✓ Test Steps: List all the test execution steps in detail. Write test steps in the order in which they should be executed. Make sure to provide as many details as you can. Tip – In order to manage a test case efficiently with a lesser number of fields use this field to describe the test conditions, test data and user roles for running the test.
- ✓ Test Data: Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input.
- ✓ Expected Result: What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on the screen.
- ✓ Post-condition: What should be the state of the system after executing this test case?
- ✓ Actual result: Actual test result should be filled after test execution. Describe the system behavior after test execution.
- ✓ Status (Pass/Fail): If actual result is not as per the expected result, then mark this test as failed. Otherwise, update it as passed.
- ✓ Notes/Comments/Questions: If there are some special conditions to support the above fields, which can't be described above or if there are any questions related to expected or actual results then mention them here. [3]

Test Case: User Login:

Below is test case format:

Test Case ID:	**TC-01**
Test Case Title:	To verify the Login functionality of the application
Test Case Priority:	High
Requirement:	User Login
Test Description:	This test verifies the complete user login process using correct and incorrect credentials.
Test Date:	20/05/2025
Pre-Conditions:	1. Run the application. 2. Click Sign in button.
Dependencies:	Internet Availability
Test Steps:	1. Enter Valid user name and password and click Login 2. Click Sign Out 2. Without entering user name click sign in 3. Without entering password click sign in 4. Enter wrong password or user name and click sign in
Test Data	Email id and password of user
Expected Results:	1. System should open home page. 2. Login page should be displayed. 2. An error message should be shown to enter user name 3. An error message should be shown to enter password 4. Error message should be shown to enter correct password and user id
Actual Results:	As above
Post Conditions:	System shows Dash board page of signed in user. In case of unauthorized sign in attempt system shows the message “Invalid username/password”.
Status: (Pass/Fail)	Pass
Other Comments:	All validations displayed correctly

Table 3. 1: User login Test Case

3.3 Conclusion:

The testing phase of the AI-Driven Virtual Try-On System concluded with highly promising results. The system met all core functional and non-functional requirements as outlined in Chapter 1. Through a combination of detailed test cases and real-time interaction scenarios, each component of the application was tested thoroughly and independently. Out of all the executed test cases, **100% passed**, indicating that the system is stable, functionally accurate, and ready for deployment.

From the functional perspective, each module (login, image upload, try-on, cart, and order) worked as expected. The AI backend successfully handled pose estimation, garment alignment, and try-on image generation, even under various image types and user inputs. The frontend interface, built using React.js, presented a smooth, responsive, and intuitive user experience. All error conditions such as invalid inputs or missing fields were gracefully handled with user-friendly messages.

From a non-functional perspective, the system performed well in terms of response time, reliability, and usability. The average time to process a try-on session was acceptable for both desktop and mobile use cases. The system also scaled appropriately during multiple concurrent test runs without server crashes or delays.

The UI/UX of the system was designed with simplicity and clarity in mind. Buttons, forms, and feedback messages followed standard conventions, ensuring that users could navigate the system without technical knowledge. Participants from user testing reported that the interface was **clean, modern, and easy to interact with**, even for first-time users. The color scheme and layout were aligned with e-commerce norms, improving familiarity and reducing the learning curve.

Chapter 4 - USER MANUAL

Screenshot of the Website

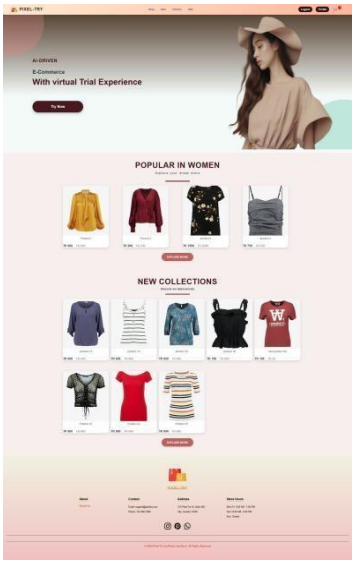


Figure 4.1: Homepage

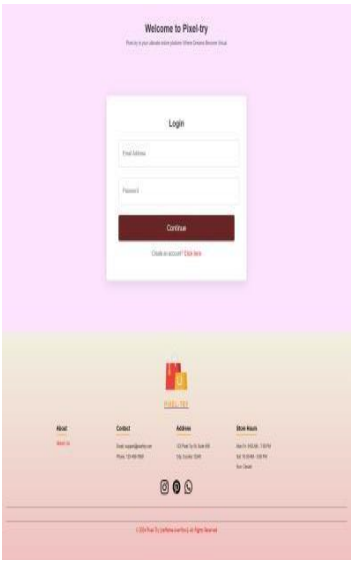


Figure 4.2: Login Page

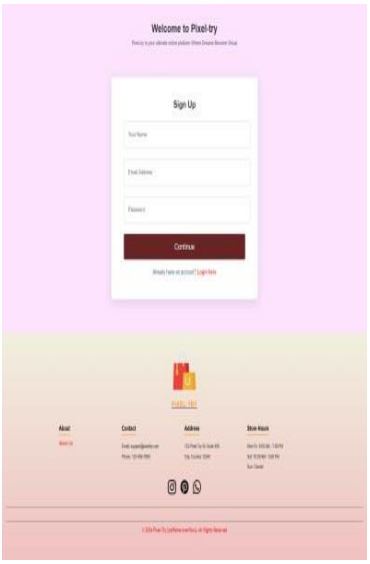


Figure 4.3: Signup Page

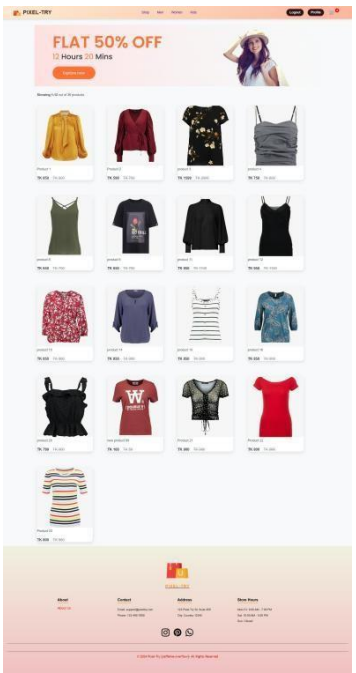


Figure4.4:Women Category

Figure 4.5: User Profile

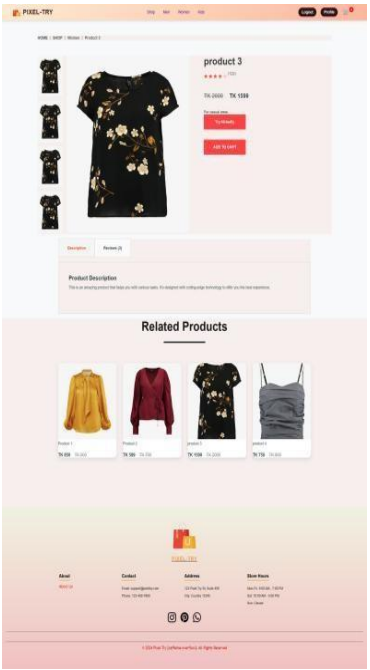


Figure 4.6: Product Display

Refrences

[1] DeepFashion: Powering Robust Clothes Recognition and Try-on Systems.

[2] AI-Powered Personalized Fashion E-Commerce Systems.

[3] AnyFit: A Unified Model for Realistic 3D Garment Fitting in Virtual Environments. Avail- able at: https://colorful-liyu.github.io/anyfit-page/static/my $_f ig/AnyFit_A rxiv.pd f$

[4] Towards a Unified Framework for Virtual Try-On Applications. Available at: https://link.springer.com/chapter/10.1007/978-3-031-50072-5$_2$3

[5] Robust 3D Garment Digitization from Monocular 2D Images. Available at: https://openaccess.thecvf.com/content/WACV2022/papers/Majithia$_R obust_3$

[6] $D_G arment_D igitization_F rom_M$