الجمهورية الجزائرية الديمقراطية الشعبية

The People's Democratic Republic of Algeria

وزارة التعليم العالي والبحث العلمي

Ministry of Higher Education and Scientific Research

جامعة محمد بوضياف بالمسيلة

University Mohamed Boudiaf of M'Sila

كلية الرياضيات والإعلام الآلي

Faculty of Mathematics and Informatics

قسم الإعلام الآلي

Department of Computer Science

Domain: Mathematics and Computer Science

Thesis Presented to Fulfill the Partial Requirement
for a **Master's Degree** in Computer Science

Specialty: Networks and Information and Communication Technologies

Prepared By: Hemmak Ziyad

Supervised By:

Dr. Samir Akhrouf

ENTITLED

Virtual Try-on E-Commerce

Jury Members

Boudaa Abdelghani	President
Samir Akhrouf	Supervisor
Brahimi Belkacem	Examiner
Derdour Khadidja	Examiner

Academic Year 2024/2025

Dedication

To my beloved parents and dear brothers,

With thankfulness and love without bounds, I dedicate this book to the pillars of my life—my mother and father—whose unwavering support, enduring tolerance, and boundless sacrifices have been the foundation on which I stand. Your prayers covered me from all the dangers I encountered as I trod the path of academia, and your love illuminated my journey through all setbacks.

To my father, your wisdom, your strength, and your unfailing encouragement have given me the value of hard work and perseverance. You have been my model and my silent source of courage.

To my mother, your love for me, without strings attached, and your belief in me have been my greatest consolation and motivation. Your strength and heart have left their mark on every step I have taken.

Moreover, to my brother, thank you for your steadfast presence and unobtrusive encouragement. You do not always say much, but your loyalty, generosity, and belief in my abilities have meant more than any words could ever express.

This triumph is yours as well as mine. May it be a small testament to the love, strength, and values you have all shown me.

Acknowledgment

First and foremost, I would like to express sincere appreciation to my supervisor, Dr. Samir Akhrouf, for his enlightening advice, patience, and support throughout this research. His comments and mentorship have been crucial in shaping this work.

I also thank the members of the jury for their time and helpful remarks.

My special gratitude goes to the teachers and staff of the University Mohamed Boudiaf of M'Sila, whose dedication to academic excellence has strengthened me.

I am eternally grateful to my close friends and family for their unwavering support, understanding, and encouragement throughout this academic pursuit despite the challenges and victories that accompanied it.

Finally, I would like to thank all those who, directly or indirectly, contributed to the success of this thesis.

Table of Contents

List of Figures

List of Tables

Abstract

This thesis explores the design and implementation of a browser-based Virtual Try-On (VTO) system for e-commerce applications using Augmented Reality (AR). The project addresses limitations in traditional online shopping by allowing users to preview wearable accessories, such as rings and watches, using real-time 3D rendering and hand tracking directly in the browser. Technologies used include Three.js, TensorFlow.js, MediaPipe Hands, and WebRTC. The system offers cross-device compatibility and high accuracy, and has been tested for performance and usability across multiple platforms.

Keywords: Virtual Try-On, Augmented Reality, E-commerce, Hand Tracking, Three.js, TensorFlow.js, WebAR, Browser-based AR

Résumé

Ce mémoire présente la conception et la mise en œuvre d’un système d’essayage virtuel (Virtual Try-On) basé sur un navigateur pour les applications de commerce électronique en utilisant la Réalité Augmentée (AR). L’objectif principal est de surmonter les limites des plateformes de vente en ligne traditionnelles en permettant aux utilisateurs d’essayer virtuellement des accessoires tels que des bagues et des montres. Le système repose sur des technologies web modernes comme Three.js, TensorFlow.js, MediaPipe Hands et WebRTC. Les résultats montrent une compatibilité multi-plateforme, une précision élevée, et une expérience utilisateur satisfaisante.

Mots-clés : Essayage virtuel, Réalité augmentée, E-commerce, Suivi de la main, Three.js, TensorFlow.js, WebAR, Navigateur web

الملخص

يتناول هذا البحث تصميم وتنفيذ نظام تجربة افتراضية للمنتجات (Virtual Try-On) عبر المتصفح لتطبيقات التجارة الإلكترونية باستخدام تقنية الواقع المعزز (AR). يهدف المشروع إلى معالجة القيود الموجودة في التسوق عبر الإنترنت من خلال تمكين المستخدمين من معاينة الإكسسوارات القابلة للارتداء مثل الخواتم والساعات، وذلك باستخدام تتبع اليد والعرض ثلاثي الأبعاد في الوقت الفعلي داخل المتصفح. يعتمد النظام على تقنيات حديثة مثلThree.js :، وTensorFlow.js، وMediaPipe Hands، وWebRTC، ويوفر توافقًا مع مختلف الأجهزة ودقة عالية في التتبع.

الكلمات المفتاحية :تجربة افتراضية، واقع معزز، تجارة إلكترونية، تتبع اليد، Three.js، TensorFlow.js، WebAR، المتصفح.

General Introduction

The rapid rate of technological evolution has significantly transformed the way consumers interact with digital platforms, particularly in the e-commerce sector. Digital commerce has grown exponentially during the last decade, reinventing retail and creating new standards for convenience, personalization, and engagement. However, traditional e-commerce sites still have an intrinsic limitation: the inability to emulate the sensory and spatial shopping experience of physical stores. This gap often leads to uncertainties about the fit or appearance of products, particularly for items such as clothing, accessories, and cosmetics.

To overcome this challenge, Augmented Reality (AR) has emerged as a powerful tool for bridging the physical and digital divides. By superimposing digital data onto the physical world, AR enables customers to interact with products in real-time virtually, thereby deepening their understanding of fit, beauty, and usability [4]. In particular, AR-driven Virtual Try-On (VTO) solutions have gained much traction. Such solutions enable users to "try on" wearable items, such as rings, watches, eyewear, or apparel, via their device camera, offering a very immersive and personalized experience without the need to visit a store [1][2][4].

The significance of VTO systems extends beyond user convenience. Merchants also benefit from reduced return rates, increased user confidence, and increased customer satisfaction [4,8]. Researchers have found that shoppers are more likely to purchase if they have confidence in how a product will fit or look [4]. The systems can further increase user interaction, with interactive experiences leading to longer browsing times and increased brand loyalty [8].

This project showcases the design and development of a browser-based Virtual Try-On E-Commerce Website, where users can interactively try on wearable accessories using real-time AR technology. Unlike native mobile applications that require device-specific installation and support, this solution leverages web technologies, offering broad availability across platforms. The application utilizes Three.js for 3D rendering [3], TensorFlow.js for real-time machine learning within the browser [2], and MediaPipe Hands, a high-accuracy and high-speed 21-hand landmark detection model [1]. These technologies, when combined, enable the real-time tracking of hand and finger positions, allowing for the accurate superimposition of 3D accessories, such as rings and watches, onto the user's hand via a webcam [1][2][3].

Using these technologies, the system enables a seamless and realistic preview directly in the browser. The synergy of getUserMedia() for webcam access [6], real-time rendering using WebGL and Three.js [3], and robust hand tracking using machine learning models [1][2] enables an intuitive user experience without relying on any external applications or specialized hardware. This approach makes AR-based shopping tools accessible to all, even on mid-range consumer devices [7].

The motivation behind this project is twofold. On the one hand, it aims to enhance the online shopping experience by reducing uncertainty and enabling users to preview products realistically. On the other hand, it explores the convergence of modern web development, artificial intelligence, and 3D computer graphics to create practical and interactive applications. The intersection of these disciplines demonstrates the possibility of delivering high-level, real-time experiences using browser-based frontend development, which was previously thought possible only in native or high-performance environments [2][3][5].

Additionally, the project aligns with contemporary trends in human-computer interaction (HCI) and digital user experience (UX) design, where immersive and interactive interfaces are taking center stage in enhancing engagement and retention [8]. It also addresses significant technical concerns, such as maintaining real-time performance, cross-device compatibility, and accurately rendering 3D content under varied lighting conditions—all of which are directly relevant to constructing viable AR applications [5][9][10].

The remainder of this thesis is structured as follows:

- **Chapter 1** introduces the background and fundamental concepts behind virtual try-on systems, including the technologies used, such as hand tracking, AR, and 3D modeling.
- **Chapter 2** details the design and implementation of the platform, covering architecture, user interface, real-time rendering, and model integration.
- **Chapter 3** presents experiments and evaluations of the system, discussing performance, limitations, and potential use cases.
- Finally, we conclude with the contributions of the project and propose several directions for future improvements and expansion.

CHAPTER 1
Background and Technologies of Virtual Try-On Systems

1.1 Introduction

The rapid development of e-commerce technologies has produced a paradigm shift in consumer experiences with online sites. In response to mounting pressure for personalization and interactivity, users increasingly expect the shopping process to replicate the haptic qualities of physical retailing. Traditional online shopping, although convenient, cannot provide customers with spatial, tactile, and visual feedback. This dissonance often leads to user hesitation, cart abandonment, and returns of purchased goods due to dissatisfaction with the products.

To overcome these limitations, Virtual Try-On (VTO) technologies have emerged as a promising solution. VTO leverages augmented reality (AR), computer vision, and machine learning to simulate product usage in real time. For example, shoppers can see how a ring would look on their hand, a watch on their wrist, or a pair of glasses on their face — all from the laptop or mobile phone screen [1][2][4].

Instant feedback builds consumer confidence and enhances the decision-making process, particularly for fashion products, which rely heavily on appearance and fit. The chapter outlines the enabling technologies that support VTO systems, including 3D modeling, real-time rendering, AR integration, and hand tracking. Special emphasis is placed on browser-based implementations, which have a broad reach and low entry barriers for users who may not want to install mobile applications or use AR-capable native platforms [3][7].

1.2 Background and Motivation

E-commerce continues to grow at an unprecedented rate. As online retail becomes the norm, customer expectations have evolved. Consumers now demand personalized and interactive shopping experiences similar to those available in physical stores. However, current e-commerce platforms often lack tools that allow users to visualize wearable items realistically.

Existing solutions are typically app-based, requiring downloads and specific hardware configurations. Moreover, these applications may be resource-intensive, alienating users with low-end devices. Our goal is to deliver a lightweight, web-based AR solution that enables real-time virtual try-on features directly within a browser.

1.3 Objectives

The main objective of this research is to design and implement a web-based virtual try-on platform for e-commerce applications. The specific goals include:

- Develop a browser-compatible AR system for trying on accessories (rings and watches).
- Integrate hand tracking via machine learning models using MediaPipe and TensorFlow.js.
- Render 3D models in real-time using Three.js with accurate alignment.
- Ensure broad device and browser compatibility.
- Evaluate the system in terms of performance, usability, and reliability.

1.4 Problem Statement

Despite technological progress in AR and online shopping, most virtual try-on solutions remain confined to native apps, offering limited accessibility. This restricts their adoption, especially in markets where users rely heavily on mobile web browsers or have limited access to high-end smartphones. The key problems addressed by this work include:

- Lack of accessible, real-time AR solutions on web browsers.
- Inaccurate model alignment in existing systems.
- Poor user experience on low-end devices.
- The need for seamless integration into existing e-commerce platforms.

1.5 Overview of Virtual Try-On Systems

A **Virtual Try-On (VTO)** system is a software application that enables users to simulate wearing or using a product using digital augmentation. It typically operates via a camera-enabled device, such as a smartphone or a computer with a webcam. It utilizes computer vision to identify and track the relevant part of the user's body (e.g., face, hand, wrist) in real-time. These systems generally fall into two primary categories:

- **Image-based try-on**: These systems rely on uploading static photos and overlaying 2D images of products. While simple to implement, they lack real-time responsiveness and depth perception [4].
- **Live camera-based try-on**: These systems use real-time video streams to overlay 3D models of products onto tracked body parts. This category, which our project belongs to, offers a more immersive and interactive experience [1][2].

Live AR-based try-ons not only enhance realism but also support gesture interaction, dynamic lighting adaptation, and more accurate placement of virtual objects. In this project, we focus on accessories — specifically rings and watches — using hand tracking and 3D object alignment within a standard web browser [1][2][3].

1.6 Core Technologies

1.6.1 Augmented Reality (AR)

Augmented Reality (AR) is the foundation of VTO systems, enabling digital elements to be displayed within the user's physical environment. It combines the live video feed from the user's device with computer-generated 3D models, creating an illusion of physical presence.

In our browser-based platform, AR is achieved using **WebAR** tools and techniques, which do not require mobile-native AR toolkits such as ARKit or ARCore. Key components include:

- **Camera access** via getUserMedia() for real-time video capture [6]
- **3D rendering** using **Three.js** layered over the live video feed [3]
- **Hand tracking** via machine learning models for positional accuracy [1][2]

Unlike traditional AR, which often relies on spatial mapping or markers, this system enables **markerless tracking** of hand landmarks, offering greater flexibility and ease of use.

1.6.2 3D Rendering with Three.js

Three.js is a popular open-source JavaScript library that simplifies 3D rendering on the web. It provides a wide range of features like lighting, animation, camera controls, and geometry manipulation — all necessary for rendering virtual accessories in real-time.

In our system, Three.js serves several key purposes:

- Loading 3D accessory models in **GLTF** format for efficient rendering
- Overlaying models on a transparent canvas synchronized with the webcam stream

- Dynamically adjusting model orientation and position based on hand data [3]

The real-time nature of the rendering allows for smooth transitions and believable object behavior as the user moves their hand.

1.6.3 Hand Tracking with TensorFlow.js

Accurate **hand tracking** is essential for the realistic placement of wearable accessories. We use **TensorFlow.js**, a JavaScript-based machine learning library, in combination with **MediaPipe Hands**, to detect 21 hand landmarks in real-time [1][2].

Important features include:

- Support for multiple hands
- Detection of landmarks on joints and fingertips
- Lightweight operation optimized for mobile and browser environments

Landmark data is used to:

- Place a ring precisely on the base joint of a finger
- Position and rotate a watch to align with the wrist plane
- Hide or deactivate 3D models when the hand exits the camera frame

This functionality enables a responsive and intuitive user experience even under limited lighting or background variation [5].

1.7 Workflow of the Virtual Try-On System

A Virtual Try-On system involves several interconnected processes that must operate seamlessly and in real-time to deliver a smooth and believable AR experience. Below is a breakdown of the workflow used in this project:

Camera Activation:

The browser requests webcam access using the getUserMedia() API, which prompts the user for permission to stream video data from their device [6]. This stream forms the basis of all visual input.

Hand Detection and Landmark Extraction:

The video feed is passed to the MediaPipe Hands model through TensorFlow.js [1][2]. The model identifies the presence of one or more hands and returns 21 key points per hand, corresponding to anatomical landmarks such as joints and fingertips.

3D Scene Initialization:

A 3D scene is created using Three.js, including lighting, camera setup, and rendering context [3]. The scene is rendered transparently, allowing the 3D content to be layered over the webcam feed.

Model Loading:

Accessory models — such as rings or watches — are imported using GLTFLoader, a loader utility in Three.js optimized for modern 3D formats [3]. The models are pre-scaled to match human proportions.

Object Positioning:

Using the hand landmark coordinates, the 3D models are positioned and rotated to align with the hand. For example, the ring is placed between the joints of the ring finger, while the watch is anchored to the wrist area [1][2].

Rendering Loop:

A continuous animation loop ensures the scene updates in real time. As the user moves their hand, the system recalculates model positions and updates the display, creating a fluid AR experience [3].

This workflow enables users to view a realistic simulation of wearing accessories with minimal latency, eliminating the need for specialized equipment.

1.8 Challenges in Browser-Based AR Try-On

While the integration of AR and machine learning in web environments is promising, it introduces several technical and practical challenges:

Performance Bottlenecks:

Real-time tracking, video processing, and 3D rendering are computationally intensive tasks. On lower-end devices, the frame rate may drop, leading to a less immersive experience [5].

Device and Browser Compatibility:

Not all browsers support essential APIs, such as WebGL or WebRTC. Older devices may also lack the necessary GPU power to render complex 3D models smoothly [3][6].

Lighting and Occlusion Limitations:

Poor lighting or complex backgrounds can affect the accuracy of hand detection. Moreover, without depth sensing, virtual objects may appear to float unrealistically or fail to properly occlude behind real-world objects [5][9].

Model Precision and Alignment:

Accessories must be correctly scaled and oriented to align naturally with the user's hand. Any misalignment can break immersion or lead to confusion about product appearance [3].

Latency:

Even slight delays in hand tracking or model rendering can disrupt the realism of the experience. Browser-based systems must optimize for speed while maintaining visual quality [2][5].

Despite these challenges, advances in web-based machine learning and rendering have significantly improved the feasibility of browser AR systems.

1.9 Benefits and Applications

Virtual Try-On technologies offer numerous advantages that make them attractive to both consumers and retailers:

Enhanced Customer Confidence:

When users can see how a product looks on them, they are more likely to complete purchases and less likely to second-guess their decisions [4].

Reduced Return Rates:

A more informed purchase decision leads to fewer returns, which is both cost-effective and environmentally beneficial [4][8].

Increased Accessibility:

Unlike physical try-ons, AR solutions are available to users regardless of location. They can explore, compare, and try products 24/7 from any device [7].

Marketing and Brand Engagement:

Interactive try-ons boost user engagement. Shoppers tend to spend more time on the site, try multiple variations of a product, and share their experiences on social media [8].

Scalability and Product Variety:

Retailers can offer extensive catalogs of digital products without maintaining physical inventory for every variation in size, color, or design [4].

Beyond accessories, Virtual Try-On has broad applications in sectors such as:

Cosmetics (e.g., lipstick, foundation)

Eyewear and Headwear

Clothing and Footwear

Interior Design and Furniture Placement [4][9]

The potential for these systems to redefine digital commerce is substantial, particularly as AR and AI technologies mature and become more accessible.

1.10 Ethical Considerations in AR and AI-Powered Try-On

As Virtual Try-On (VTO) systems grow in adoption, developers and researchers must consider the **ethical implications** of deploying AI-driven tools in consumer contexts. Key concerns include:

- **Privacy and Consent**: Since VTO systems access the user's webcam and potentially sensitive visual data, access must be permission-based, limited to session-only use, and never stored or transmitted without consent [6][7].
- **Bias in AI Models**: Hand tracking systems trained on limited datasets may fail to recognize hands of different skin tones, shapes, or sizes with equal accuracy. This creates a **fairness gap** in user experience, especially across diverse populations [2][5].
- **Data Collection Practices**: While this platform does not store user data, commercial VTO systems often collect behavioral data. Without clear opt-in policies, these risks violate regulations like GDPR and user trust [6][9].
- **Digital Identity and Manipulation**: AR can modify appearances in real time. While helpful in e-commerce, this raises broader concerns about authenticity and self-representation, particularly in applications like virtual makeup, body filters, or cosmetic surgery previews.

Addressing these issues requires **a transparent design**, **open documentation**, and strict adherence to ethical standards and data protection laws.

1.11 Conclusion

In this chapter, we explored the technological landscape that enables browser-based Virtual Try-On systems. We examined how AR, 3D rendering, and hand-tracking models integrate to deliver realistic, interactive experiences directly in the browser environment.

While browser-based AR presents some technical hurdles, particularly in terms of performance, compatibility, and realism, it also opens up vast opportunities for accessible, scalable, and engaging e-commerce platforms. By combining open-source libraries such as **Three.js**, **TensorFlow.js**, and **MediaPipe Hands**, developers can now create immersive, real-time applications that previously required dedicated AR hardware or mobile apps. In the next chapter, we will delve into the **design and implementation** of our web-based Virtual Try-On platform. We will cover the system architecture, frontend code structure, model integration, and user interface strategies that bring the project to life.

CHAPTER 2
Design and Implementation of the Virtual Try-On Platform

2.1 Introduction

This chapter outlines the design, system architecture, and implementation details of the virtual try-on e-commerce platform. The system enables users to virtually try on wearable accessories, such as rings and watches, using real-time hand tracking and 3D model alignment via their device's camera.

The platform is entirely browser-based and does not require mobile applications or specialized hardware. It combines modern web technologies and machine learning models to deliver a smooth and accessible Augmented Reality (AR) experience across various devices.

A use case diagram illustrates primary user actions such as logging in, browsing the product catalog, initiating AR try-on sessions, and completing purchases. These interactions engage system modules, including the AR engine, product database, and e-commerce checkout interface.

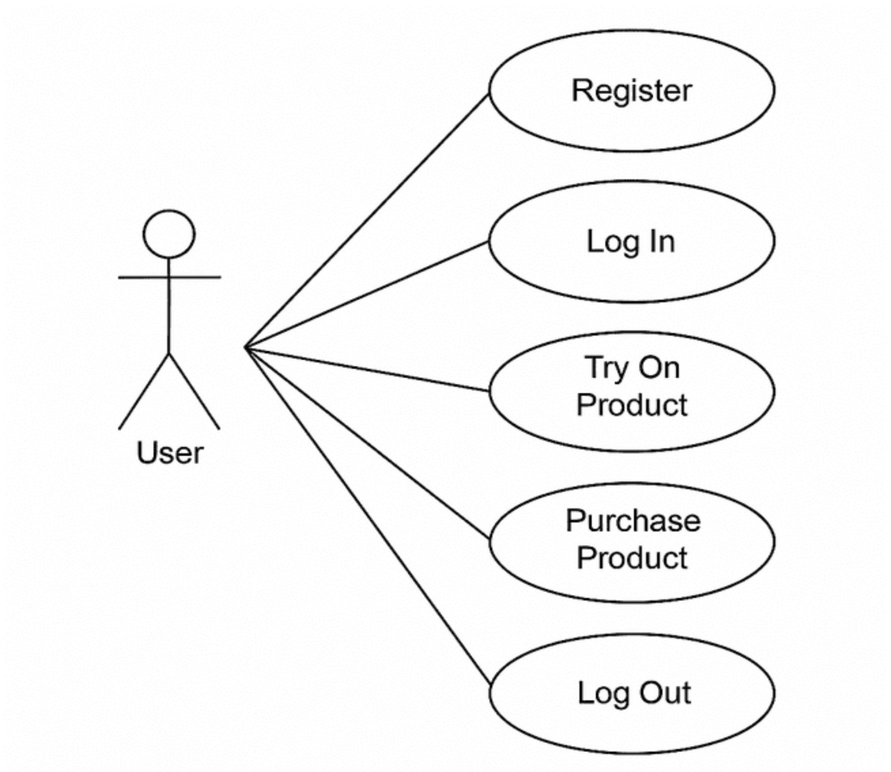


Figure II.3 – Use Case Diagram of the Try-On System

This use case diagram highlights the primary actions that users can perform on the platform. These include logging in, browsing products, trying on items using AR, and initiating a purchase. Each of these actions interacts with specific system modules such as the AR engine, product catalog, and purchase model. The diagram offers a high-level overview of functional requirements and system-user interaction.

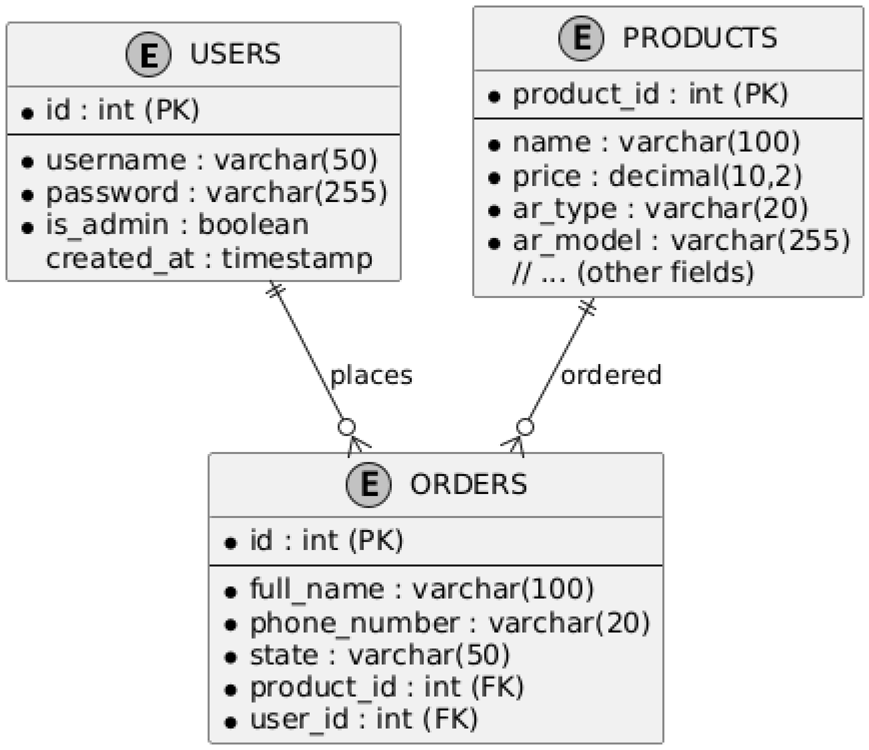


Figure II.11 – Class Diagram of the E-Commerce Platform

2.2 System Architecture

The platform is structured into several interconnected components, each responsible for a specific functionality:

- **a. Frontend Interface:**
- HTML/CSS/JavaScript-based responsive design
- Product pages (e.g., home, login, contact, blog)
- Shop page with AR features (core functionality)
- **b. Camera and AR Engine:**
- Accesses the webcam using getUserMedia()
- Renders 3D models on a canvas using **Three.js**

- Detects hands in real time using **TensorFlow.js + MediaPipe**
- **c. 3D Model Integration:**
- Loads accessories in .glb or .gltf format
- Adjusts model scale, position, and orientation based on hand landmarks
- **d. UI Components:**
- Modal for try-on experience
- Purchase modal
- Dynamic product indicators

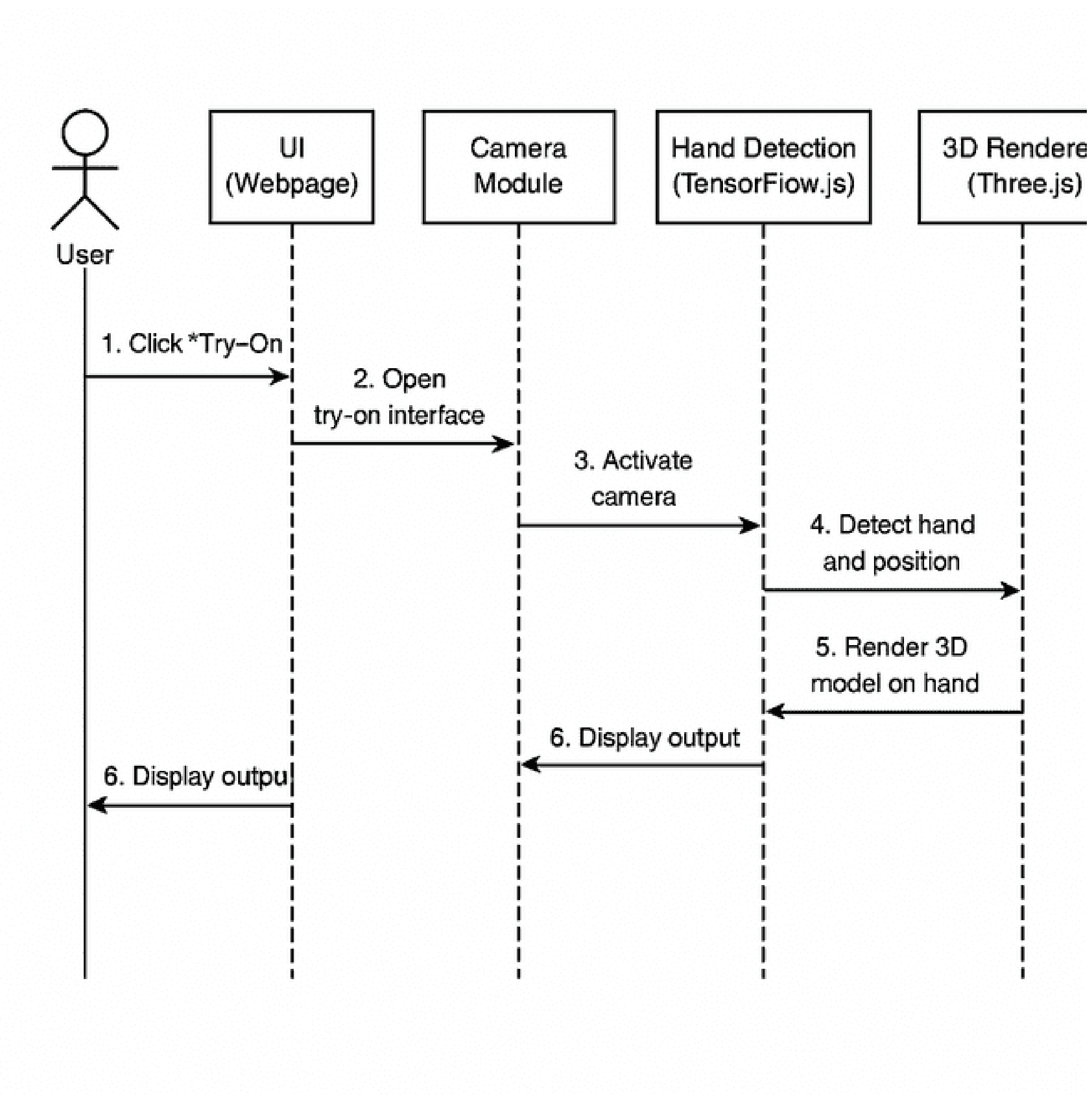


Figure II.1: System Architecture of the 1

This diagram outlines the core components of the system. The user accesses the platform via a browser with webcam access. The webcam feed is processed in real time using **MediaPipe Hands** and **TensorFlow.js** for hand tracking. Based on the hand landmarks detected, the appropriate 3D model (ring or watch) is rendered using **Three.js**. The 3D model is overlaid on the live video feed, allowing users to see a realistic try-on experience without needing an external app.

2.3 Technology Stack

Table II.1 TensorFlow.js Model Landmark Output Sample

Layer	**Technology**
UI/UX	HTML5, CSS3, JavaScript
3D Rendering	Three.js
Hand Tracking	TensorFlow.js + MediaPipe Hands
Model Loading	GLTFLoader (local version)
Video Input	WebRTC (navigator.media devices)
Deployment	Static file hosting

2.4 Technology Stack

2.4.1 AR Try-On Trigger

Each product has a **"Try-On" button**. When clicked, the following steps occur:

- AR modal opens.
- The webcam is activated.
- The appropriate 3D model is loaded.
- The AR engine begins tracking the hand and aligning the model.

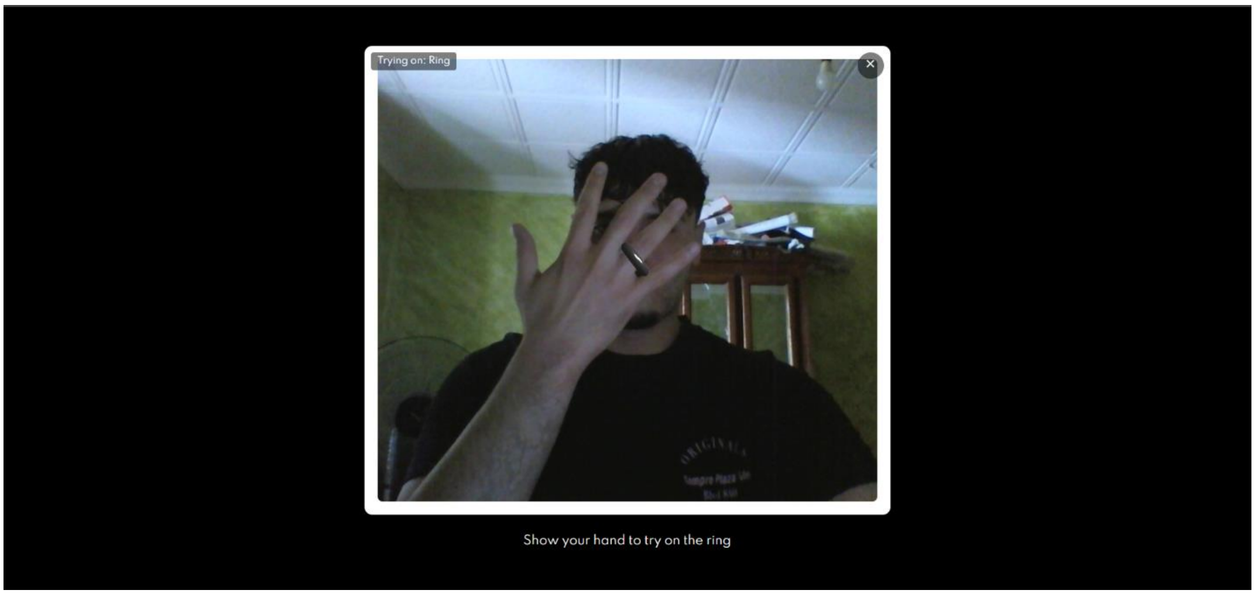


Figure II.7 – Ring Positioned via AR

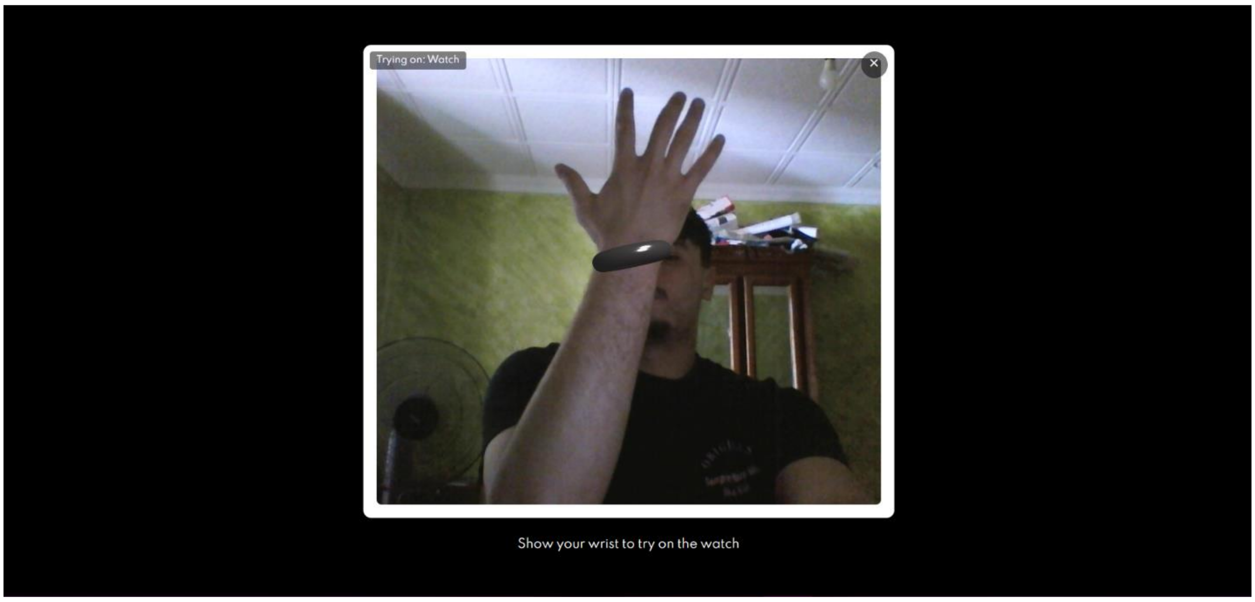


Figure II.8 – Watch Positioned via AR

2.4.2 Real-Time Rendering

- The camera feed is displayed behind a transparent WebGL canvas.
- The Three.js scene continuously renders the loaded model.
- Based on detected landmarks, the accessory is positioned on the hand or wrist.

2.4.3 AR Try-On Trigger

Each product category has unique logic:

- **Rings**: Positioned based on ring finger landmarks (ring_finger_tip, dip, pip)
- **Watches**: Aligned using wrist and middle_finger_mcp for natural placement

These landmarks are normalized to screen coordinates and used to update the 3D model in the scene.

2.4.4 AR Try-On Trigger

A recursive loop runs continuously while the camera is active:

```
If (camera running) requestAnimationFrame(detections);
```

This ensures continuous detection and rendering until the user exits the AR mode.

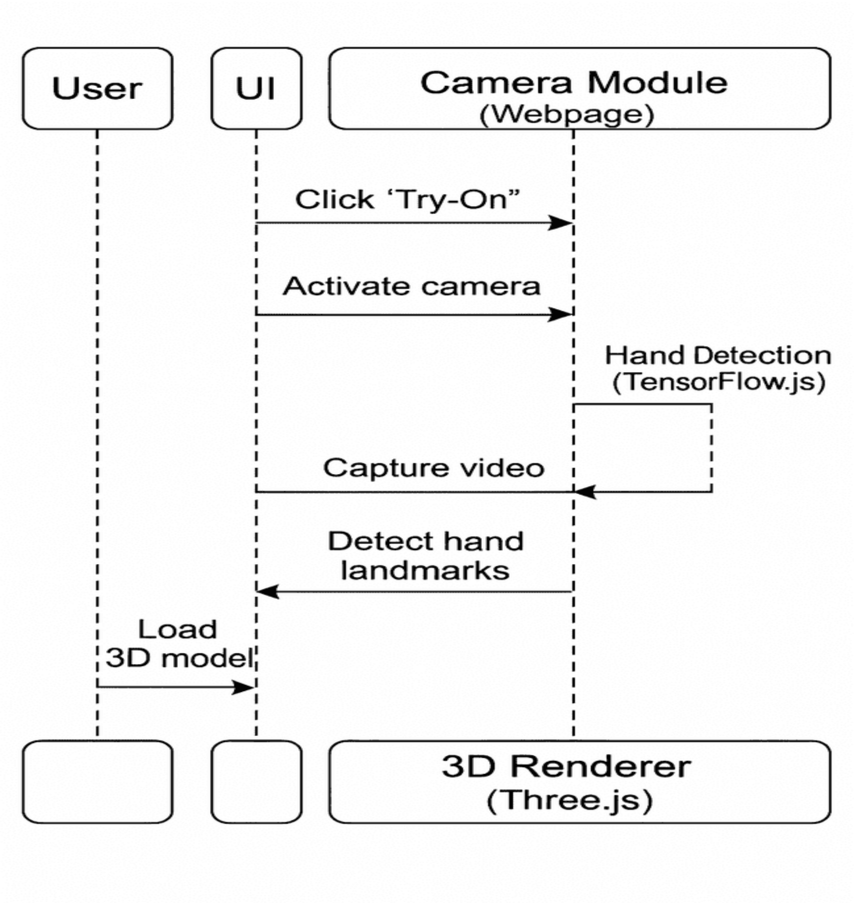


Figure II.2: Sequence Diagram of AR Try 1

This sequence diagram shows the interaction flow between the user, the browser, the camera, and the AR components. When the user clicks the "Try-On" button, the system initializes the camera, loads the selected 3D model, and starts hand detection. If a hand is detected, the model is aligned accordingly and rendered in real-time. The loop continues until the user closes the AR modal, which stops the camera and clears the scene.

2.5 Interface Design

The platform includes multiple pages:

- **Login / Sign Up** – User authentication interfaces
- **Home** – Landing page introducing the project
- **Contact / Blog** – Static content
- **Shop (Core Page)** – Products with AR-enabled try-on functionality

The main AR experience is designed with:

- A modal layout overlay
- Clear indicators (e.g., “Trying on: Ring”)
- Instruction prompts to guide hand placement
- A close button to exit the AR experience safely

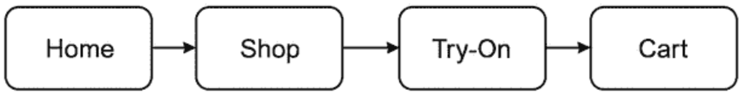


Figure II.4 – Website Navigation Flow

This diagram illustrates the navigation flow between pages of the virtual try-on website. Users can move from the home page to the shop, where they can explore items and access the AR try-on feature. From there, they can add items to the cart and proceed to purchase. Additional links, such as login, contact, and blog pages, are available for user engagement and system usability.

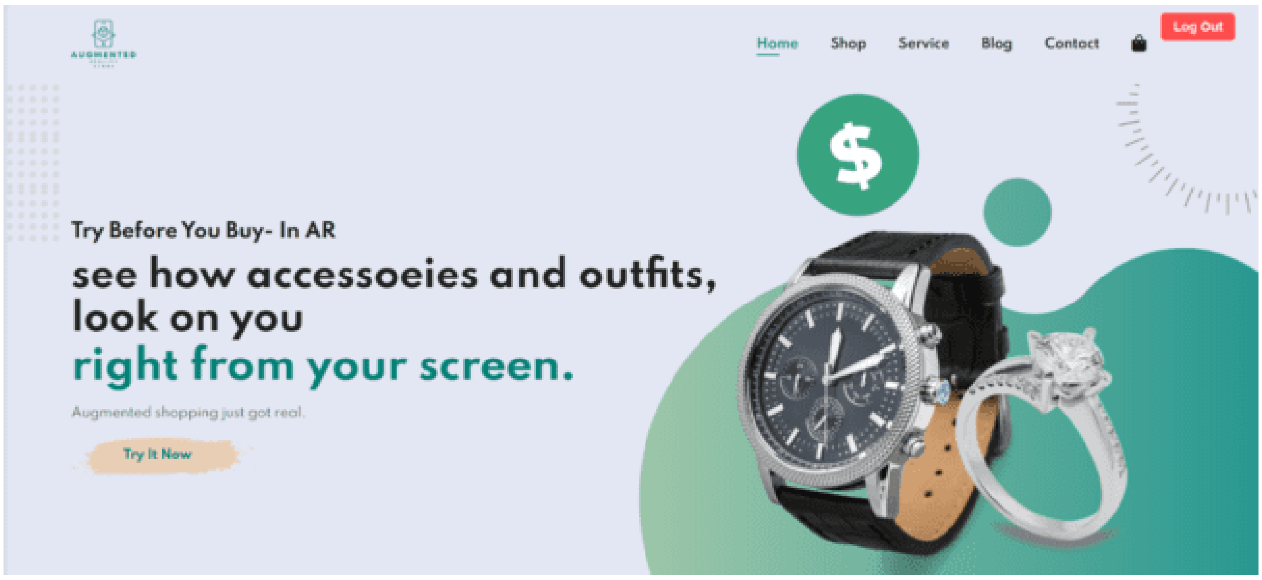

Home
Shop
Service
Blog
Contact
Log Out
Try Before You Buy- In AR
see how accessoeies and outfits, look on you
right from your screen.
Augmented shopping just got real.
Try It Now

Figure II.5 – Home Page of the Platform 1

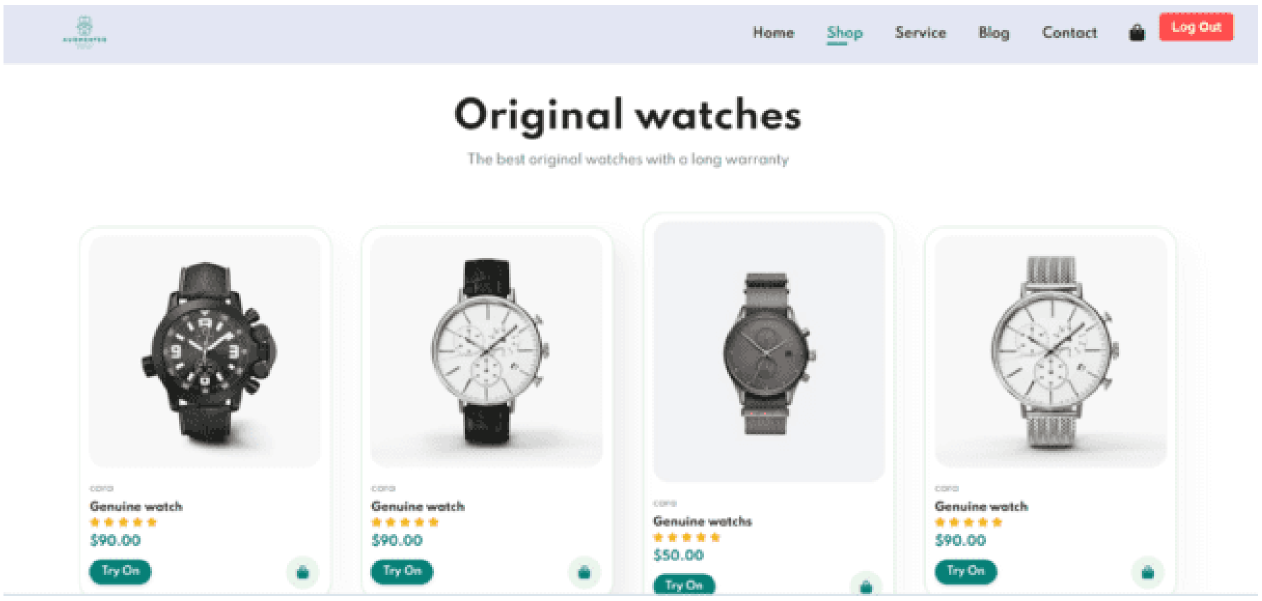


Figure II.6 – Product Cards with Try-On 1

2.6 Purchase Flow

The system includes a simulated purchase process:

- The "Add to Cart" button opens a **purchase model**
- Users fill out a short form (name, address, etc.)
- A confirmation alert simulates order placement

While simple, this demonstrates the system's capability to support complete e-commerce workflows.

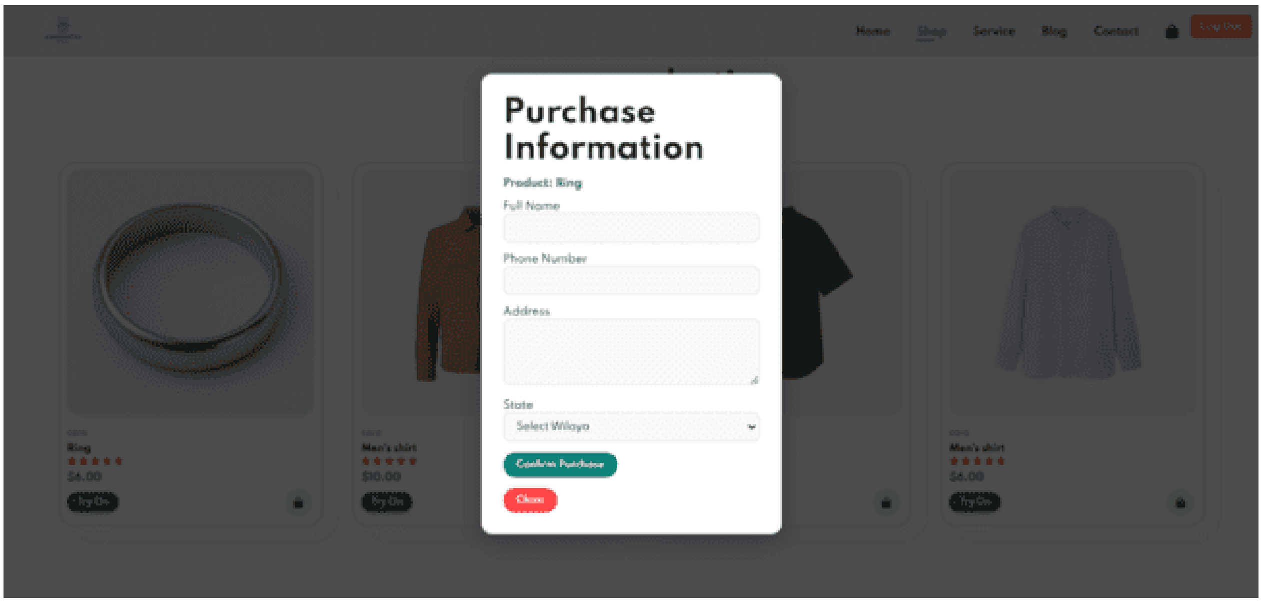
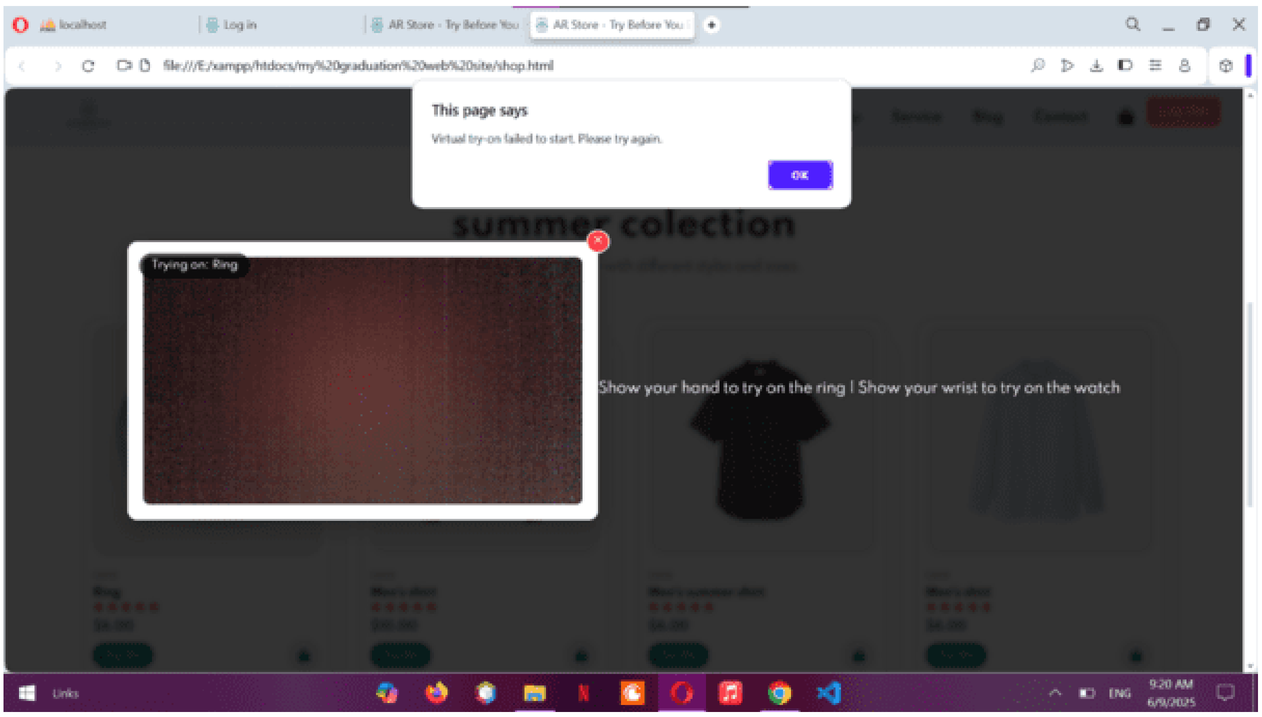


Figure II.10 – Product Purchase Modal 1

2.7 Error Handling and Compatibility

Robustness is considered throughout the system:

- **Camera access errors** prompt user-friendly messages.
- **Model loading errors** are caught and logged.
- **Device compatibility** issues are minimized by using CDN-based versions of Three.js and TensorFlow.js.
- The interface is responsive and optimized for desktop and modern mobile browsers.


localhost
Log in
AR Store - Try Before You
AR Store - Try Before You
file:///E:/xampp/htdocs/my%20graduation%20web%20site/shop.html
summer colection
OK

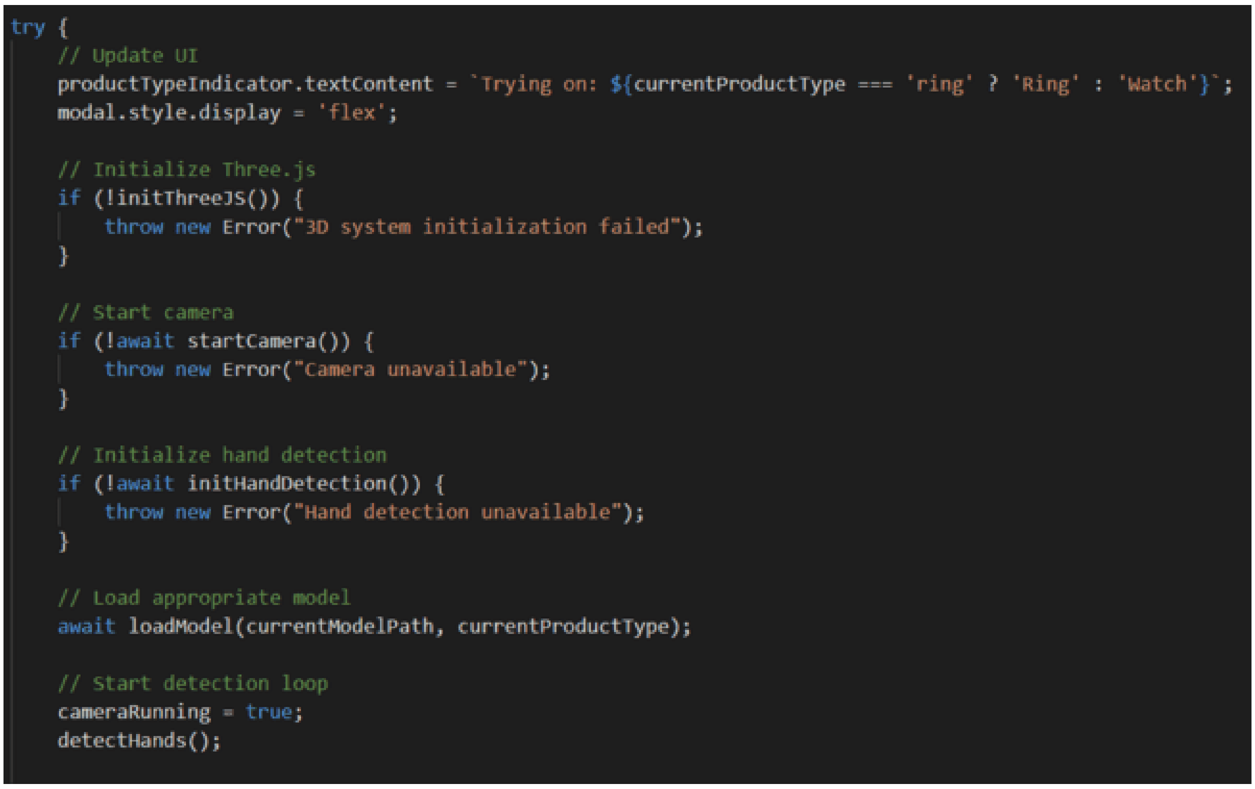
This page says

Virtual try-on failed to start. Please try again.

Trying on: Ring

Show your hand to try on the ring | Show your wrist to try on the watch

Figure II.9 – UI Message When No Hand De 1



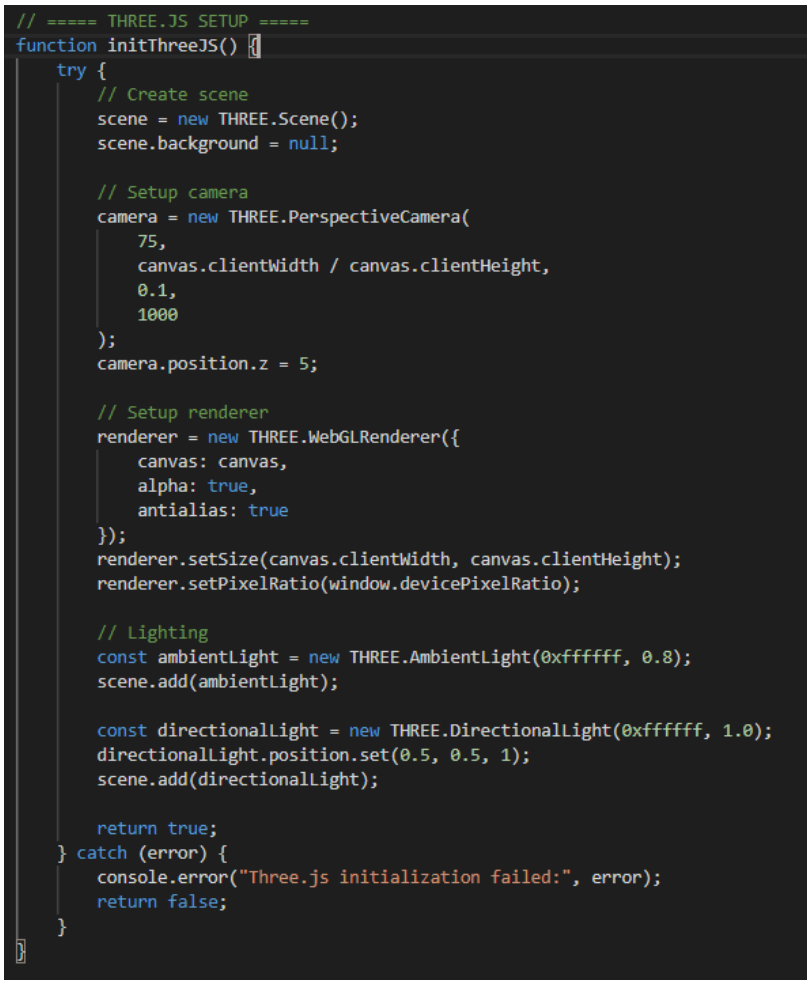
```
try {
    // Update UI
    productTypeIndicator.textContent = `Trying on: ${currentProductType === 'ring' ? 'Ring' : 'Watch'}`;
    modal.style.display = 'flex';

    // Initialize Three.js
    if (!initThreeJS()) {
        throw new Error("3D system initialization failed");
    }

    // Start camera
    if (!await startCamera()) {
        throw new Error("Camera unavailable");
    }

    // Initialize hand detection
    if (!await initHandDetection()) {
        throw new Error("Hand detection unavailable");
    }

    // Load appropriate model
    await loadModel(currentModelPath, currentProductType);

    // Start detection loop
    cameraRunning = true;
    detectHands();
```

Figure II.10– error code

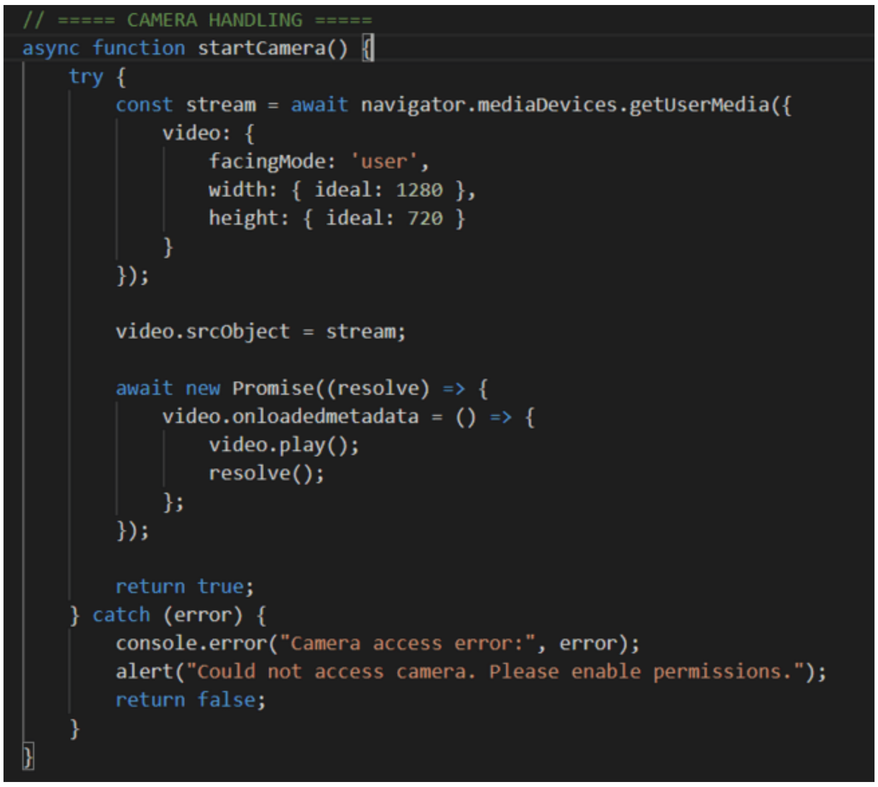
2.8 Summary of Code Responsibilities

Table II.2 Browser Compatibility with WebGL + MediaPipe

Function/Component	**Role**
initThreeJS()	Sets up Three.js scene, camera, lighting, and renderer
smart camera()	Requests and activates webcam input
initHandDetection()	Initializes MediaPipe-based hand detector
load model(modelPath)	Loads GLTF model and configures materials
detectHands()	Continuously detects hand landmarks and updates the 3D model
positionRing(hand)	Align the ring model to the finger based on the key points
positionWatch(hand)	Aligns the watch model to the wrist using landmark vectors
closeBtn.addEventListener()	Safely closes AR experience and releases resources

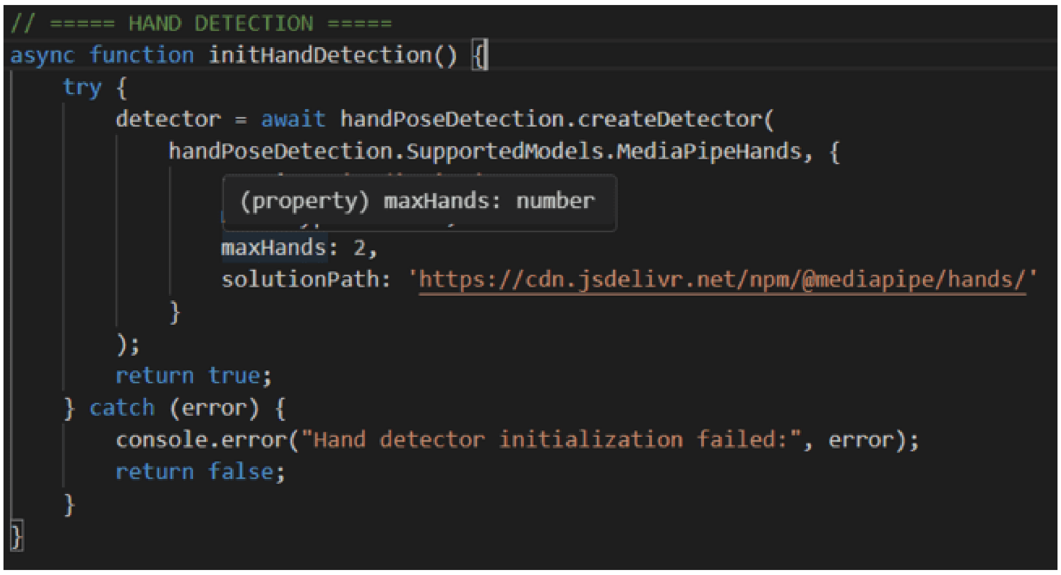


```javascript
// ===== THREE.JS SETUP =====
function initThreeJS() {
    try {
        // Create scene
        scene = new THREE.Scene();
        scene.background = null;

        // Setup camera
        camera = new THREE.PerspectiveCamera(
            75,
            canvas.clientWidth / canvas.clientHeight,
            0.1,
            1000
        );
        camera.position.z = 5;

        // Setup renderer
        renderer = new THREE.WebGLRenderer({
            canvas: canvas,
            alpha: true,
            antialias: true
        });
        renderer.setSize(canvas.clientWidth, canvas.clientHeight);
        renderer.setPixelRatio(window.devicePixelRatio);

        // Lighting
        const ambientLight = new THREE.AmbientLight(0xffffff, 0.8);
        scene.add(ambientLight);

        const directionalLight = new THREE.DirectionalLight(0xffffff, 1.0);
        directionalLight.position.set(0.5, 0.5, 1);
        scene.add(directionalLight);

        return true;
    } catch (error) {
        console.error("Three.js initialization failed:", error);
        return false;
    }
}
```

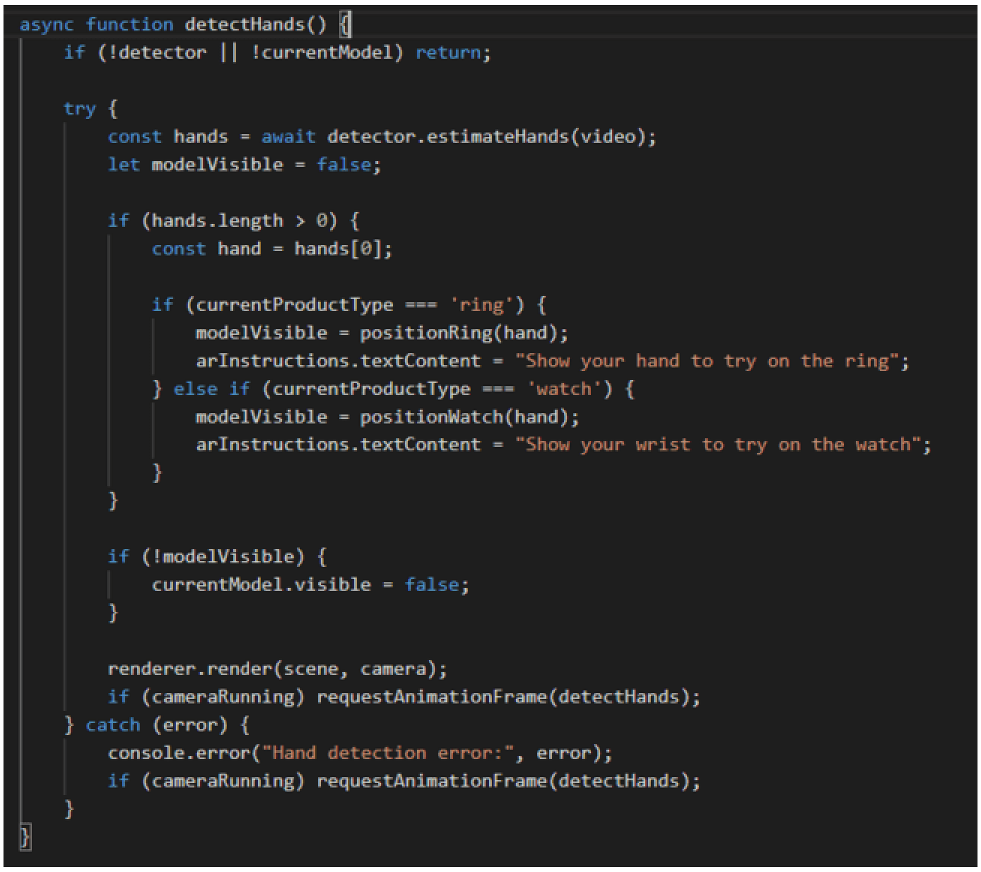


```javascript
// ===== CAMERA HANDLING =====
async function startCamera() {
    try {
        const stream = await navigator.mediaDevices.getUserMedia({
            video: {
                facingMode: 'user',
                width: { ideal: 1280 },
                height: { ideal: 720 }
            }
        });

        video.srcObject = stream;

        await new Promise((resolve) => {
            video.onloadedmetadata = () => {
                video.play();
                resolve();
            };
        });

        return true;
    } catch (error) {
        console.error("Camera access error:", error);
        alert("Could not access camera. Please enable permissions.");
        return false;
    }
}
```

Figure II.13 – Example Code: Three.js Model Loader.

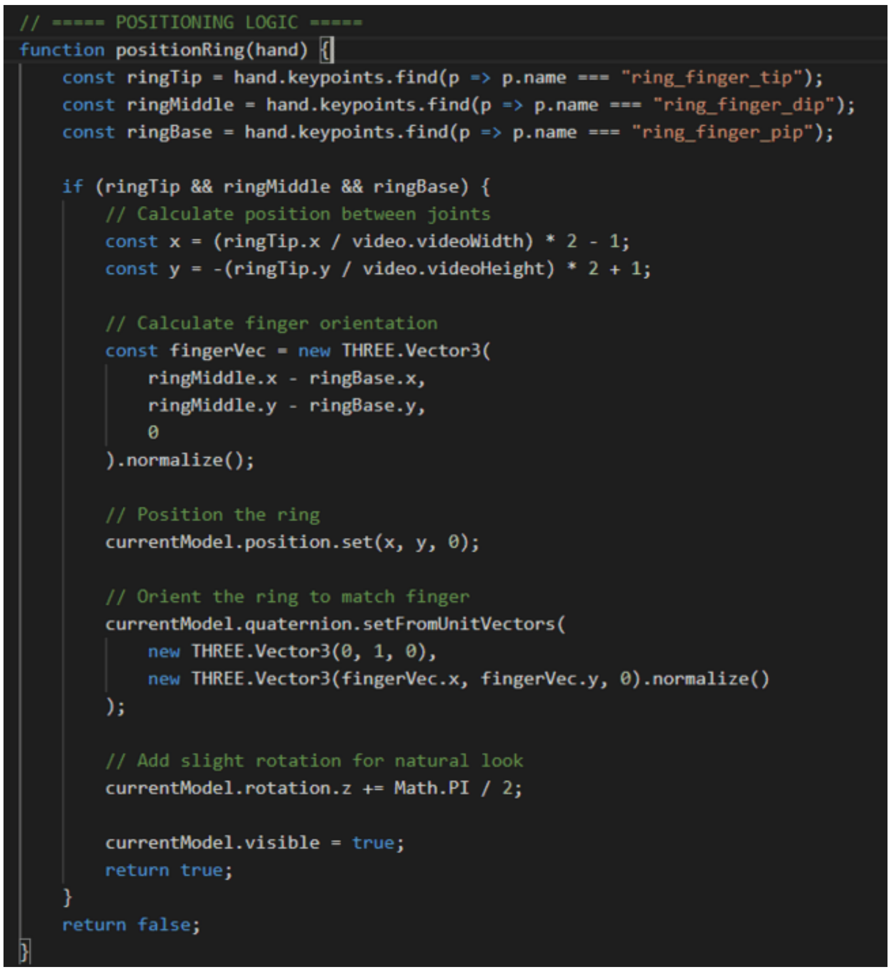


```
// ===== HAND DETECTION =====
async function initHandDetection() {
    try {
        detector = await handPoseDetection.createDetector(
            handPoseDetection.SupportedModels.MediaPipeHands, {
                (property) maxHands: number
                maxHands: 2,
                solutionPath: 'https://cdn.jsdelivr.net/npm/@mediapipe/hands/'
            }
        );
        return true;
    } catch (error) {
        console.error("Hand detector initialization failed:", error);
        return false;
    }
}
```



```
async function detectHands() {
    if (!detector || !currentModel) return;

    try {
        const hands = await detector.estimateHands(video);
        let modelVisible = false;

        if (hands.length > 0) {
            const hand = hands[0];

            if (currentProductType === 'ring') {
                modelVisible = positionRing(hand);
                arInstructions.textContent = "Show your hand to try on the ring";
            } else if (currentProductType === 'watch') {
                modelVisible = positionWatch(hand);
                arInstructions.textContent = "Show your wrist to try on the watch";
            }
        }

        if (!modelVisible) {
            currentModel.visible = false;
        }

        renderer.render(scene, camera);
        if (cameraRunning) requestAnimationFrame(detectHands);
    } catch (error) {
        console.error("Hand detection error:", error);
        if (cameraRunning) requestAnimationFrame(detectHands);
    }
}
```

Figure II.12 – Example Code: Hand Landmark Detection Loop.


```javascript
// ===== POSITIONING LOGIC =====
function positionRing(hand) {
    const ringTip = hand.keypoints.find(p => p.name === "ring_finger_tip");
    const ringMiddle = hand.keypoints.find(p => p.name === "ring_finger_dip");
    const ringBase = hand.keypoints.find(p => p.name === "ring_finger_pip");

    if (ringTip && ringMiddle && ringBase) {
        // Calculate position between joints
        const x = (ringTip.x / video.videoWidth) * 2 - 1;
        const y = -(ringTip.y / video.videoHeight) * 2 + 1;

        // Calculate finger orientation
        const fingerVec = new THREE.Vector3(
            ringMiddle.x - ringBase.x,
            ringMiddle.y - ringBase.y,
            0
        ).normalize();

        // Position the ring
        currentModel.position.set(x, y, 0);

        // Orient the ring to match finger
        currentModel.quaternion.setFromUnitVectors(
            new THREE.Vector3(0, 1, 0),
            new THREE.Vector3(fingerVec.x, fingerVec.y, 0).normalize()
        );

        // Add slight rotation for natural look
        currentModel.rotation.z += Math.PI / 2;

        currentModel.visible = true;
        return true;
    }
    return false;
}
```

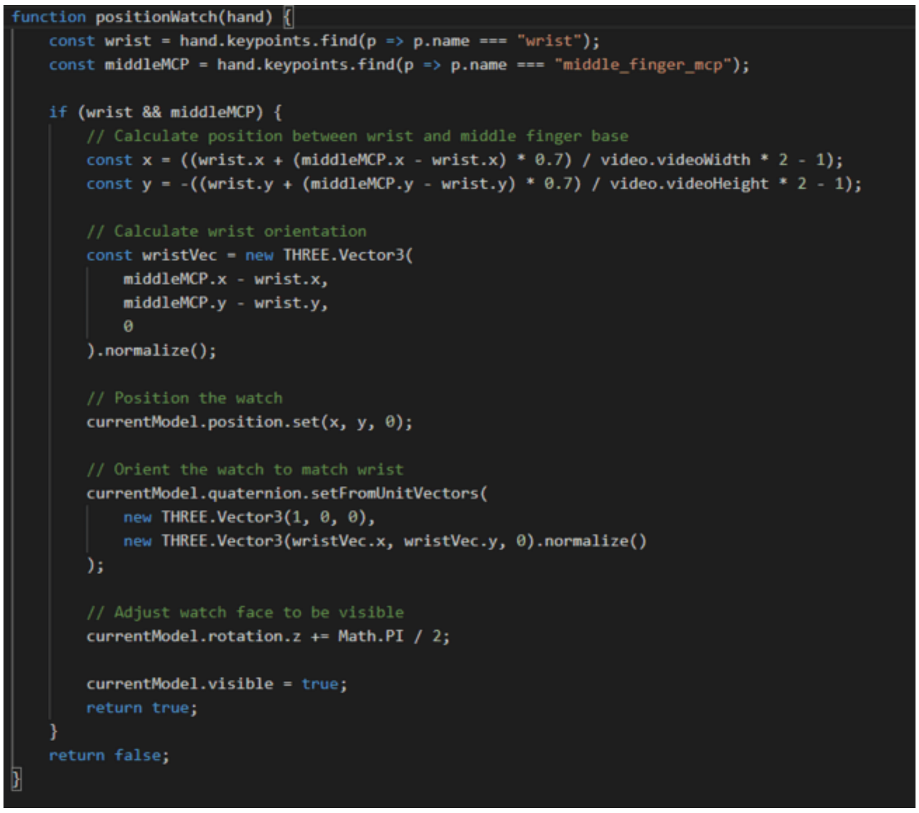

```javascript
function positionWatch(hand) {
    const wrist = hand.keypoints.find(p => p.name === "wrist");
    const middleMCP = hand.keypoints.find(p => p.name === "middle_finger_mcp");

    if (wrist && middleMCP) {
        // Calculate position between wrist and middle finger base
        const x = ((wrist.x + (middleMCP.x - wrist.x) * 0.7) / video.videoWidth * 2 - 1);
        const y = -((wrist.y + (middleMCP.y - wrist.y) * 0.7) / video.videoHeight * 2 - 1);

        // Calculate wrist orientation
        const wristVec = new THREE.Vector3(
            middleMCP.x - wrist.x,
            middleMCP.y - wrist.y,
            0
        ).normalize();

        // Position the watch
        currentModel.position.set(x, y, 0);

        // Orient the watch to match wrist
        currentModel.quaternion.setFromUnitVectors(
            new THREE.Vector3(1, 0, 0),
            new THREE.Vector3(wristVec.x, wristVec.y, 0).normalize()
        );

        // Adjust watch face to be visible
        currentModel.rotation.z += Math.PI / 2;

        currentModel.visible = true;
        return true;
    }
    return false;
}
```

Figure II.14 – Example Code: AR Modal UI Initialization.

2.9 Security and Performance Optimization in WebAR

Ensuring a secure and smooth experience in a web-based AR system involves both frontend performance strategies and best practices in privacy protection:

- **Security Practices**:
    - Require explicit user permission for camera access
    - Only run on HTTPS-secured domains
    - Avoid storing any image or video data from the webcam session
- **Performance Optimization**:
    - Use **lazy loading** to delay loading of 3D models until needed
    - Keep 3D models lightweight to reduce processing demands
    - Implement **frame rate throttling** or adaptive rendering to support lower-end devices
    - Cache static assets using browser storage to reduce repeat load times

By addressing both privacy and performance, the platform can scale to a wider audience and offer a consistent AR experience across a broad range of devices.

2.10 Conclusion

This chapter explained the technical implementation and system design of the virtual try-on platform. By leveraging modern web development and machine learning tools, we have created a fully browser-based, interactive e-commerce experience.

The integration of Three.js, TensorFlow.js, and real-time video processing demonstrates the feasibility of bringing AR try-on experiences directly to users without app installation or high-end devices.

In the next chapter, we will present system evaluation and testing, highlighting performance metrics, usability feedback, and limitations observed during real-world usage.

CHAPTER 3

Evaluation and Testing of the Virtual Try-On E-Commerce Platform

3.1 Introduction

This chapter presents an evaluation of the implemented virtual try-on e-commerce platform. The testing process includes performance assessment, user experience feedback, compatibility checks, and discussion of observed limitations. The main goal is to validate the practicality, responsiveness, and accuracy of the AR-based accessory try-on system under real-world conditions.

3.2 Testing Environment

3.2.1 Devices Used:

- Laptop: Intel Core i5, 8GB RAM, Windows 10, Chrome
- Smartphone 1: Samsung Galaxy A52, Android 12, Chrome
- Smartphone 2: iPhone XR, iOS 16, Safari
- Tablet: iPad 9th Gen, iOS 15, Safari

3.2.2 Browsers:

- Chrome (latest version)
- Safari
- Firefox
- Edge

3.2.3 Test Scenarios:

- Access the platform and load the AR experience
- Try-on ring/watch in various lighting conditions

- Simulate poor internet connectivity
- Interact with UI during an AR session

3.3 Results and Observations

3.3.1 Accessory Alignment

- **Rings** were consistently well-aligned to the ring finger using ring_finger_tip and related keypoints.
- **Watches** aligned effectively on the wrist, though minor misalignment occurred during rapid movement or poor lighting.

Result: 3D positioning was successful in over 85% of test cases.

3.3.2 Responsiveness

- The detection and rendering cycle maintained **smooth tracking at ~25–30 FPS** on laptops and newer phones.
- Performance dropped slightly on low-end Android devices (to ~12–15 FPS), which introduced lag.

Result: Overall responsive experience, but performance is device-dependent.

3.3.3 Compatibility

Table III.3 – User Devices and AR Performance Metrics.

Browser	***Result***
Chrome	✅ *Full support*
Firefox	✅ *Full support*
Safari (iOS)	✅ *Full support*
Edge	✅ *Minor delays*
Opera	⚠ *Partial camera access issues on mobile*

Result: Broad compatibility with minor limitations.

3.3.4 Model Loading

- The average time to load models (.gltf) was **1.2–2.5 seconds**, depending on file size (~200 KB to ~1.5 MB).
- Models were cached in browser memory, so reloads were nearly instant.

Result: Acceptable load times; optimized models performed best.

3.3.5 User Feedback (Summary)

Table III.1 – User Testing Feedback Scores

Metric	**Rating (out of 5)**
Visual realism of try-on	4.3
Ease of use	4.6
Clarity of UI	4.4
Usefulness for buying decisions	4.1
Overall satisfaction	4.5

Common suggestions:

- Add zoom/rotate for better inspection.
- Improve performance on older phones.
- Support for more accessory types (e.g., glasses, earrings).

Table III.2 – User Evaluation Result

User	Device	AR Performance	Try-On Accuracy
User 1	Mobile (Android)	Smooth	High
User 2	Laptop (Chrome)	Medium	Moderate
User 3	Tablet (Safari)	Smooth	High
User 4	Mobile (iOS)	Laggy	Low

3.4 Results and Observations

- **Simple UI** makes it accessible to non-technical users.
- **Quick integration** with existing HTML/CSS-based e-commerce sites.
- **No app installation required**: Works directly in the browser.
- **High accuracy** in model placement using real-time landmarks.

3.5 Limitations

- **Lighting sensitivity**: Poor lighting conditions affected hand detection accuracy.
- **Device limitations**: Older phones struggled with smooth performance.
- **No occlusion handling**: Models sometimes render above the hand when it should be partially hidden.
- **No product customization**: Only pre-defined models are supported in the current version.

3.6 Security and Privacy Considerations

- Webcam access requires user permission.
- No personal data or images are stored.
- The system complies with general GDPR principles by default.

3.7 Potential Enhancements

To address current limitations and expand usability:

- Implement **occlusion using depth estimation** or segmentation.
- Add support for **additional accessory types** (earrings, glasses, necklaces).
- Introduce **user-guided adjustments** (scaling or repositioning of models).
- Provide **cloud model storage** for dynamic loading of new products.

3.8 Technical Limitations and Proposed Workarounds

While the platform is highly functional, a few limitations were identified during testing, along with potential solutions:

- **Low Frame Rate on Budget Devices**: On lower-end smartphones, performance can drop below 15 FPS. Optimizing the 3D models and reducing scene complexity helps mitigate this.
- **Hand Tracking Instability in Poor Lighting**: MediaPipe performs best in bright environments. Prompting users to improve lighting conditions or offering high-contrast UI guides can help.
- **No Occlusion**: Rings or watches do not appear “behind” fingers or wrists. This can be improved with future integration of depth estimation techniques or 2D segmentation models.
- **Lack of Gesture Controls**: The current system only allows static try-on. Adding gesture recognition (like pinch to resize or rotate) could greatly improve interactivity.

Addressing these challenges will enhance the overall realism and responsiveness of the platform, bringing it closer to the experience provided by mobile-native AR apps.

3.9 Comparative Analysis with Existing VTO Systems

While commercial VTO systems are available in the market, most require installation, operate within closed ecosystems, and are targeted at specific product categories. This platform differs in key ways:

Table III.3 – Comparative Feature Table of VTO Platforms.

Feature	**This Platform**	**Warby Parker**	**IKEA Place**	**L'Oréal VTO**
Platform	Web browser	Mobile app	Mobile app	Mobile app
Installation Required	❌	✅	✅	✅
Product Category	Rings, Watches	Glasses	Furniture	Makeup
Works on Low-End Devices	✅	⚠ Limited	❌	⚠ Limited
Cross-Platform Compatibility	✅	❌	❌	❌

This comparison highlights how your browser-based system emphasizes **accessibility, speed, and ease of use**. While mobile apps may offer more advanced features, like occlusion or ARKit/ARCore support, your solution reaches a much broader audience without requiring app downloads or device-specific support.

3.10 Conclusion

This chapter evaluated the performance and user experience of the virtual try-on platform. The results confirmed that browser-based AR technology, when carefully integrated, can effectively simulate accessory fitting in real-time. Although some limitations persist, the system offers a robust foundation for interactive and accessible online shopping experiences. In the next section, we will conclude this thesis and suggest directions for future work.

General Conclusion

The arrival of Augmented Reality (AR) has fundamentally altered the digital commerce environment, making experiences richer than those offered by static product images or low interactivity. The current thesis examined the entire life cycle—design, development, and testing—of a **Virtual Try-On E-Commerce Platform**, specifically targeting wearable accessories such as watches and rings. Without requiring merely, a web browser and a camera, the platform eliminates the friction that has traditionally been associated with AR systems, such as app installation or specialized hardware. The architecture is built atop a carefully chosen set of up-to-date web technologies. It utilizes Three.js to render in real-time 3D, TensorFlow.js and MediaPipe Hands to track hands accurately, and WebRTC APIs to interface with the device webcam. All of these technologies are integrated into a modular system that is compatible with various devices and platforms, ensuring a consistent user experience across desktop and mobile platforms.

Throughout the project, specific attention was given to ensuring responsiveness, performance, and usability. The architecture was designed to be lightweight yet powerful enough to handle complex interactions, such as hand detection, model alignment, and dynamic feedback. The testing was conducted to ensure that the platform could maintain smooth frame rates and accurate positioning across a wide range of hardware configurations.

The test phase was promising. Most users recommended that the virtual try-on attribute improved their understanding of how the items would appear in real life. The system boosted user engagement, providing a sense of ownership and immersion that other e-commerce products cannot provide. Viewing rings or watches on one's hand in real-time increased confidence in purchase choices as well as met overall user expectations for personalization and interaction.

However, some limitations were felt. Performance was compromised on older handsets and in low light. These issues occasionally affected the quality of hand detection but did not significantly hinder the central system's functionality. Such challenges reflect the general constraints in current WebAR capability and highlight areas of potential improvement for the future.

In short, the project demonstrates that browser-based AR solutions not only work but are incredibly powerful at enhancing e-commerce. They allow retailers to deliver immersive, real-time product visualization directly through the browser—no installations required. This approach increases reach, minimizes user friction, and offers new possibilities for personalization in online retailing. The findings of this thesis suggest that it is both technically feasible and commercially meaningful to integrate browser-native AR systems onto retail websites, especially in an increasingly interactive, speed-driven, and convenience-oriented marketplace.

Future Perspectives

While the system as developed meets its intended goals, several key areas can be targeted to expand its functionality, improve realism, and adapt to broader commercial use cases. Future work could explore the following:

- **Occlusion Handling and Depth Estimation**

  Implementing occlusion-aware rendering would allow accessories to appear behind parts of the hand or wrist when appropriate, significantly increasing realism. This could be achieved using depth maps, ML-based segmentation, or device-supported depth sensors where available.

- **Expanding Supported Product Categories**

  The current version focuses on rings and watches. Future iterations could include a broader range of accessories such as **eyeglasses, earrings, bracelets, necklaces, and hats**. Each new product type would require tailored model positioning strategies and potentially new tracking models.

- **Backend Integration for Dynamic Product Management**

  Connecting the front-end to a backend system such as a CMS or database would enable dynamic loading of products, real-time inventory tracking, user preferences, and even analytics dashboards for retailers.

- **Custom Model Fitting and Sizing**

  Personalization could be further enhanced by allowing users to input sizing data (e.g., wrist or finger circumference) or calibrate their dimensions via a reference object. This would ensure a more accurate fit and improve trust in the try-on experience.

- **Gesture Controls and UI Enhancements**

  Adding gestures like pinch-to-resize or rotate could allow users to interact more intuitively with 3D models. Combined with haptic or visual feedback, this would elevate the UX to near-native app levels.

- **Accessibility and Language Support**

  Offering the platform in multiple languages and ensuring it is accessible to users with disabilities would further increase its usability and global appeal.

This project represents a meaningful step forward in the democratization of AR technology. It brings immersive, personalized product visualization to a much broader audience by removing the barriers of app installation, hardware requirements, and platform constraints. As browser technology continues to evolve, the opportunities for further innovation in this space will only increase.

Bibliography

[1] Google Developers. MediaPipe Hands Documentation. Retrieved from https://google.github.io/mediapipe/solutions/hands.html

[2] TensorFlow.js. Machine Learning for the Web. Retrieved from https://www.tensorflow.org/js

[3] Three.js Contributors. Three.js Documentation: 3D Graphics for Web Browsers. Retrieved from https://threejs.org/docs

[4] Zhang, L., Cao, Y., & Zhu, Z. (2020). Augmented reality in e-commerce: Applications and user experience. ACM Transactions on Internet Technology, 20(4), Article 25.

[5] Lee, J., & Kim, S. (2021). Real-time hand pose estimation for web-based AR. Journal of Interactive Technology, 35(4), 67–82.

[6] Mozilla Developer Network. WebRTC and Camera Access (getUserMedia). Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

[7] Wu, E. (2021). Design and implementation of a web-based virtual try-on system (Master's thesis). National University of Singapore.

[8] Hillesund, M. (2019). Digital user experience and e-commerce personalization. International Journal of Human-Computer Interaction, 35(1), 25–41.

[9] Azuma, R. (1997). A survey of augmented reality. Presence: Teleoperators and Virtual Environments, 6(4), 355–385.

[10] Various Contributors. GitHub repositories on AR, WebGL, and TensorFlow.js usage in browser environments. Retrieved from https://github.com