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

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

Deliverable No: D5.2

Initial version of the eTryOn interaction systems

Due date of deliverable:	30 November 2020
Actual submission date:	13 December 2020
Version:	Final version of D5.2

Main Authors: Anastasios Papazoglou Chalikias (CERTH)

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

Deliverable title	Initial version of the eTryOn interaction systems
Deliverable number	D5.2
Deliverable version	Final
Contractual date of delivery	30 November 2020
Actual date of delivery	13 December 2020
Deliverable filename	eTryOn_D5.2.docx
Type of deliverable	Demonstrator
Dissemination level	PU
Number of pages	54
Work Package	WP5
Task(s)	T5.1, T5.2, T5.3
Partner responsible	CERTH
Author(s)	Orestis Sarakatsanos (CERTH), Maria Kyrou (CERTH), Anastasios Papazoglou Chalikias (CERTH)
Editor	Elisavet Chatzilari (CERTH), Panagiotis Petrantonakis (CERTH)
Reviewer(s)	Jamie Sutherland (MLZ)

Abstract	D5.2 presents the initial versions of the eTryOn interaction systems which include three interactive solutions that were thoroughly described in D5.1.
Keywords	VR, AR, Human Fashion Interaction, UI, Mockups, Implementation, Software, Development.

Copyright

Deliverable history

Version	Date	Reason	Revised by
1	12/10/2021	Table of Contents	Tasos Papazoglou Chalikias
2	03/12/2021	Initial version	Tasos Papazoglou Chalikias
3	08/12/2021	Comments by	Jamie Sutherland
4	13/12/2021	Revised version	Tasos Papazoglou Chalikias
5	13/12/2021	Final version	Elisavet Chatzilari, Panagiotis Petrantonakis

List of abbreviations and Acronyms

Abbreviation	Meaning
VR	Virtual Reality
AR	Augmented Reality
3D	Three dimensional
HDR	High Dynamic Range
UI	User Interface
FBX	FilmBox
ZIP	Compressed file(s)
JSON	JavaScript Object Notation
UV	UltraViolet
XR	Extended Reality

Table of Contents

1. Executive summary

D5.2 presents the initial versions of the eTryOn interaction systems which include three interactive solutions that were thoroughly described in D5.1 and are outlined below:

- **VR Designer**: A Virtual Reality (VR) creative fashion application targeting fashion designers, facilitating them throughout the creative process of garment design by offering realistic fitting of the digital garments on photorealistic 3D avatars,
- **Dress Me Up**: A fashion mobile application for social media users (e.g. influencers), allowing them to virtually change their outfit in an image or video by selecting from a pool of digital garments and then uploading it to social media, and
- **Magic Mirror**: A mobile-based e-commerce Augmented Reality (AR) fashion app that enables virtual try-ons of garments during online shopping that aims to recreate the at home experience of buying clothes from a physical store.

In this document we start with a brief introduction section that outlines the updates for each interaction system, then move on to document each software implementation update and further development by use case.

Finally we outline the Software Documentation for the VR Designer and Dress Me Up in the Appendix.

This deliverable will feed the pilot phase.

2. Introduction

Following up the details on each interaction system presented in D5.1 we analyze the updates in implementing each one, in the subsequent sections that follow.

More specifically, for the first use case, VR Designer we outline the scene and lighting improvements, user interface interaction improvements, desktop version changes and garment texture improvements.

Next, in the VR Designer config app section, we describe the design changes in the garment upload procedure and outline the addition of a new Market Segmentation screen.

For the Dress Me Up application we illustrate the implemented user interface which is substantially different in look and feel than the one in the design mockups, while analyzing changes in functionality.

Furthermore, for the Magic Mirror application we outline our first steps in implementation, analyzing thoroughly the loading and storing of asset bundles, creation of obi blueprints, garment rigging and stitching.

Moving on, Snapchat Lens filter is now considered a complete work and screenshots of the filter in action as well as the appropriate Snapcodes are featured in this section.

Lastly in the appendix section there is software documentation available for the VR Designer and Dress Me Up use case.

All publicly available software is licensed under Apache 2.0 license.[1]

[1] https://www.apache.org/licenses/LICENSE-2.0

3. Implementation Updates

3.1 VR Designer desktop application

The VR Designer desktop application is a virtual reality application aimed at designers to check the real time fitting of garments in a variety of bodies, referred to as avatars or mannequins, using static or dynamic poses (animations). The user will use a VR headset to be able to navigate the three-dimensional virtual space.

This application software is open source and available at https://gitlab.com/etryon/uc1/vr-designer-app.

3.1.1 Scene and Lightning Improvements

In our previous deliverable we described how the VR Designer application looks like inside the Unity game engine. Based on the received feedback from the end users (designers), we have managed to achieve various improvements regarding the overall feel of the scene, with the most visible ones regarding the general room structure and lighting.

Figure 1: Initial implementation of the VR Designer space

Figure 2: Current implementation of the VR Designer space

Some clear differences can be identified comparing **Figures 1** & **2**. First of all, we changed the wall and floor textures to more appropriate ones in order for the room to look more realistic. We also added the ceiling and ceiling lights.
As we have mentioned, our objective is to create a feel of a real designer studio.

The scene lighting has also been improved significantly.

First of all the whole room is better illuminated by utilizing realtime global illumination settings and properties. This results in a much clearer image and allows a much better inception of the garment.
Several post processing effects have been applied to the scene in order to create a more realistic feel:

- First of all a bloom effect was applied to the ceiling lights to create light diffusion around them.
- Second, we added ambient occlusion effects so that each point of the scene receives better ambient lighting and the shadowing feels more natural and with more depth.
- We also inserted color grading filters so that the colors of the scene objects look more natural and crisp.
- Lastly, we applied an HDR lighting material provided by the designers. This results in better color accuracy, better bloom effects and less banding in low light areas.

3.1.2 UI Interactions improvements

Significant changes have also been incorporated to the user interface.

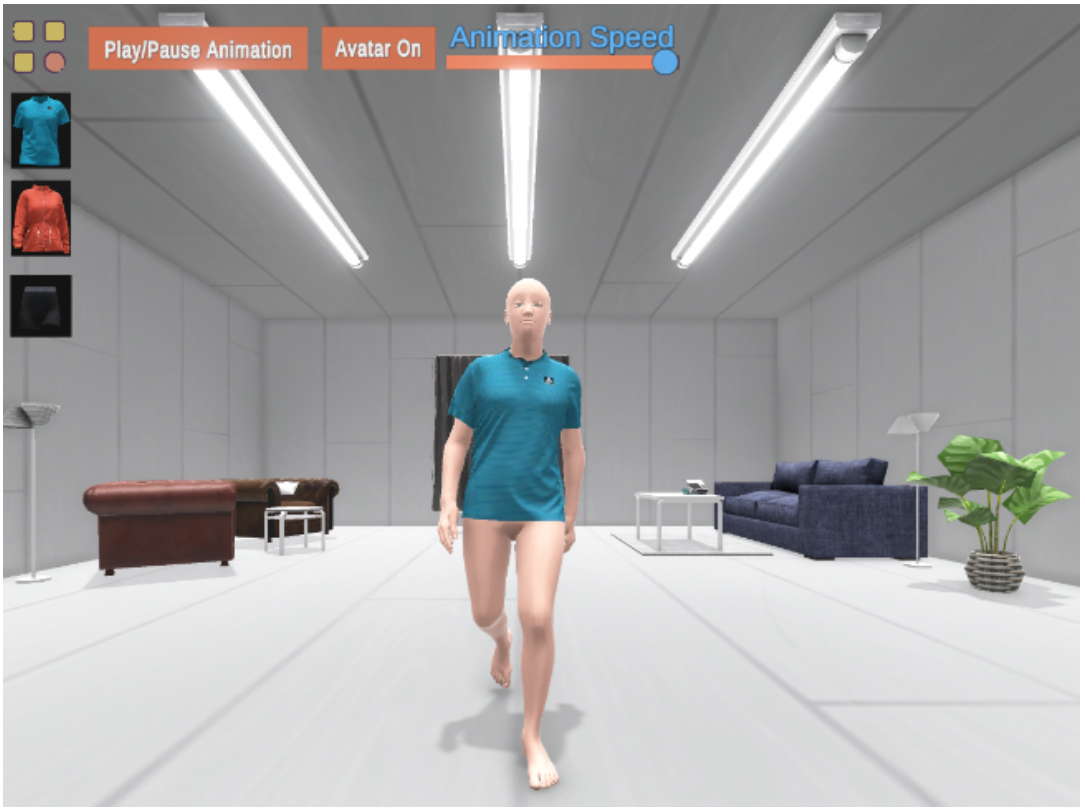

Figure 3: VR Designer app - User Interface (open)

A floating menu button has been added at the top left of the screen.

By clicking on the button, a list of all available garments appear (**Figure 3**). For each available garment, the user can toggle an animation sequence by clicking on one of the available buttons as seen in **Figure 4**. The speed of the animation can be adjusted by using the Animation slider at the top center of the screen.

After iterative testing by the ODLO designers, and based on their feedback, an option to hide the human avatar was also added (**Figure 5**). This allows them to better inspect the garment, because in some cases the avatar is considered a distraction for the designer.

Further actions to improve the UI are currently being worked on. More specifically we plan to dynamically change how the existing garments and animations appear in the menu depending on their availability on our database. The available garments and animations are subject to change and the UI needs to be able to handle such changes dynamically (by adding/removing buttons on the fly) while providing the user with the best possible experience.

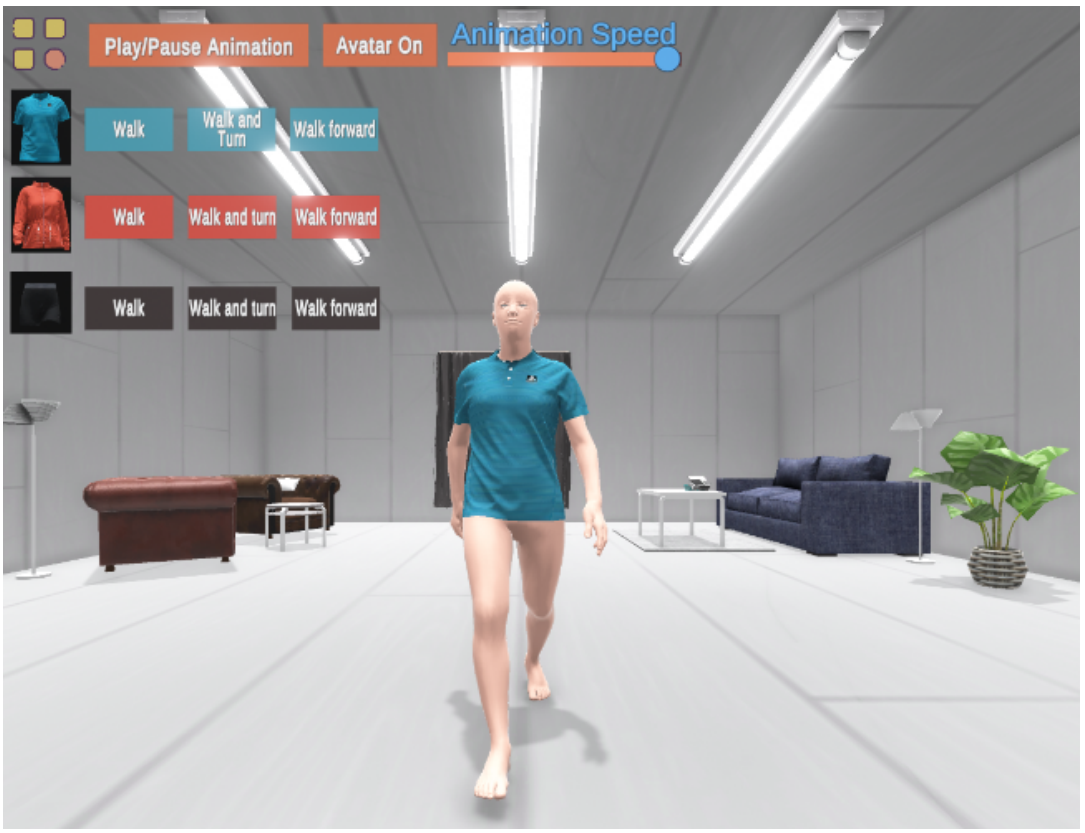

Figure 4: VR Designer app - User Interface (open with animation options)

Figure 5: VR Designer app - User Interface (garment only view)

3.1.3 Garment texture improvements

The preparation of the garments and their import in Unity was thoroughly investigated so that the garment fabrics and textures look as realistic as possible.
Through iterative testing and based on designer feedback, we changed the Shader materials for each garment part in order to correctly utilize the Specular and the normal maps provided by the designers for each garment (**Figure 6**).

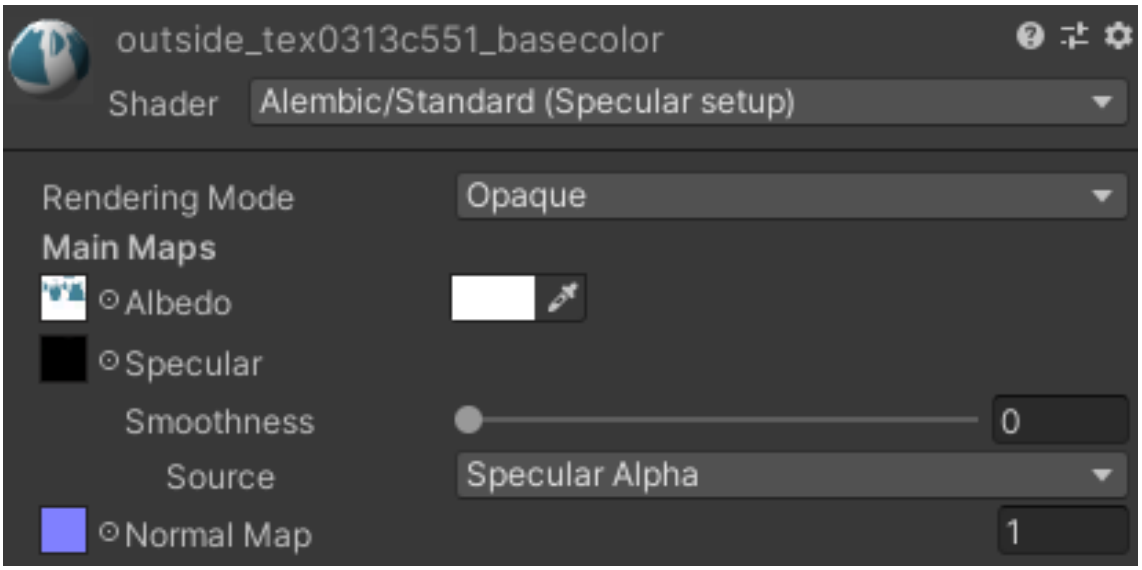

Figure 6: VR Designer app - Garment texture configuration

This indeed resulted in a significantly better and more detailed look of the garments inside Unity, however, there is still room for improvement in order to make the transition from the design software to Unity game engine more accurate.

3.2 VR Designer Config application

The VR Designer Config app is an administrative tool, where a designer can upload garments, avatars and animations in order for these files to be available inside the virtual reality space of the VR Designer application.

3.2.1 Changes in garment upload section

We further simplified the mechanism to upload a garment. Users must click on the **SELECT** button in the top right as we can see in **Figure 7**. Then they can multi-select three files needed for the garment upload. Namely:

- The garment FBX file,
- A zip that contains all the textures of the garment and
- A JSON file that contains metadata of the garment.

The platform checks that all needed files are correct and in the supported format, and then populates the '**Available Colors**' section automatically.

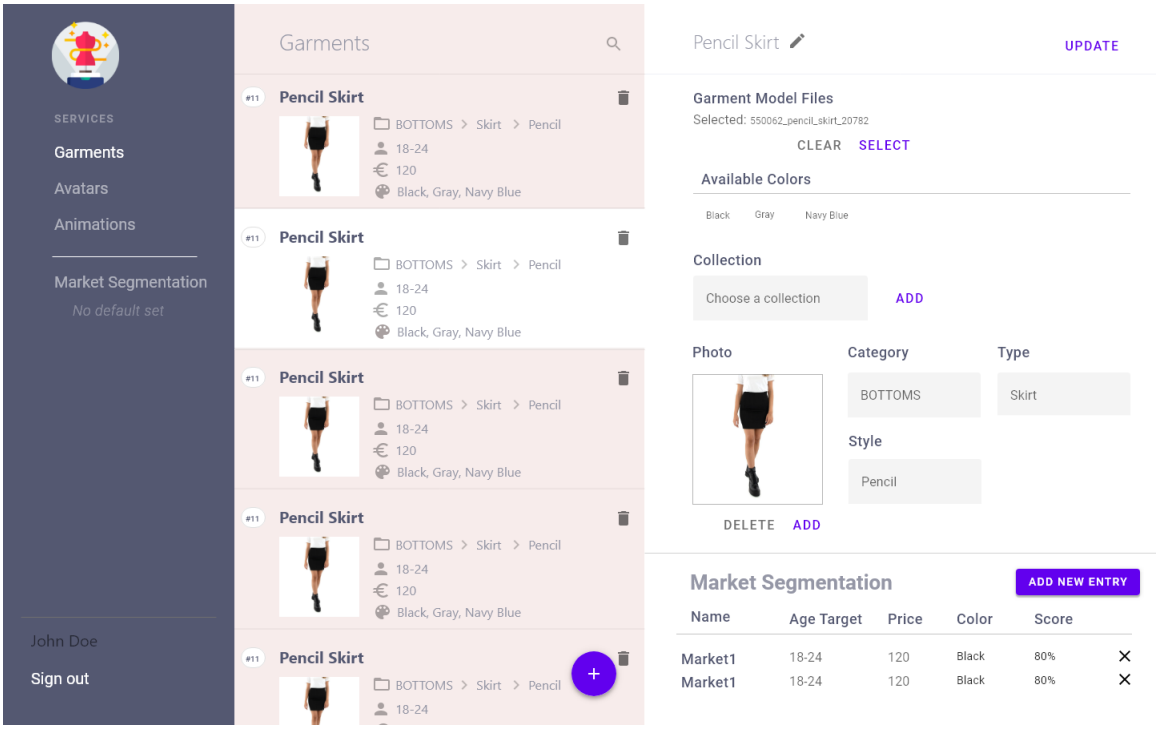

Figure 7: VR Designer Config app - Garments section

Next, right below is the **Collection** section. The user can add tags that characterize this specific garment, for example 'Spring/Summer Collection' etc.

The other sections of the garment management remain the same. The changes outlined above are also present in the **Add a new garment** screen as seen in **Figure 8**.

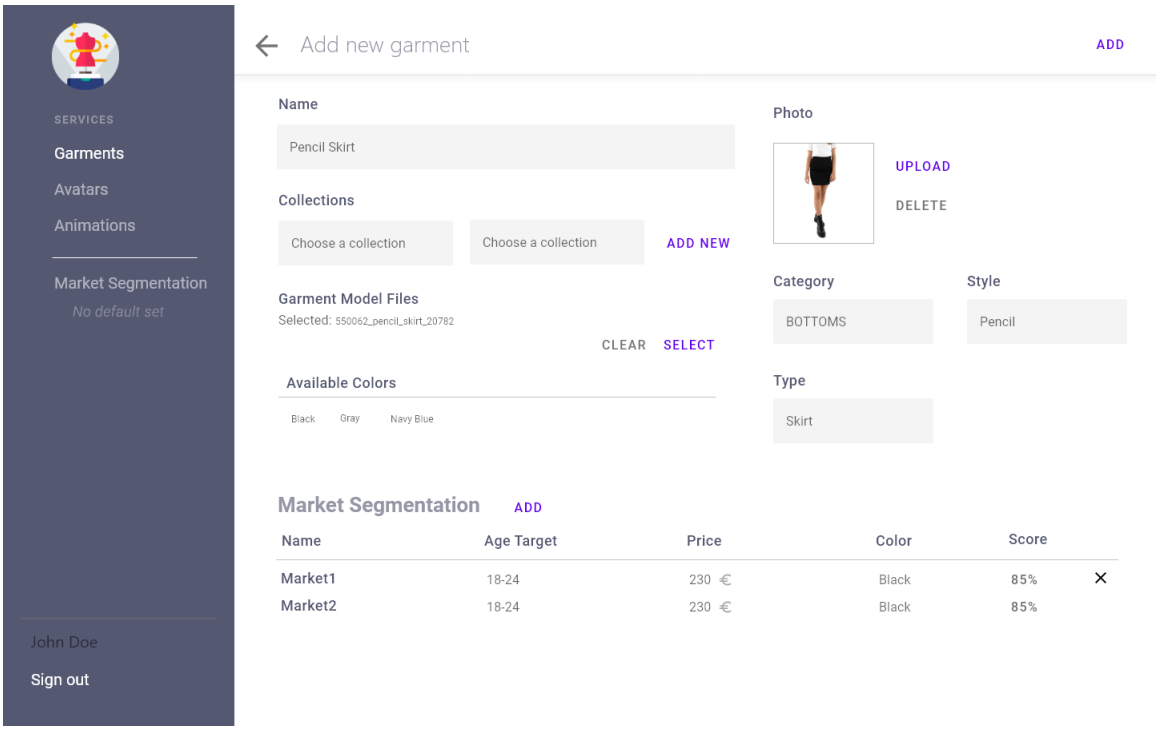

Figure 8: VR Designer Config app - Add a new garment section

3.2.2 Market Segmentation screen addition

In order for the administrators to easily manage Market Segment data for the garments, we added a new screen that features a list of the available market segments as well as a way to add or edit entries (**Figure 9**).

As we described in *D5.1 - Core version of the eTryOn interaction systems*[2], each garment has some information available like Age Target range, price and colour that is fed to Mallzees performance score service, which returns a percentage as a prediction of the trend detection classifier developed in WP3.

Users can add many entries featuring varying values for the three aforementioned fields, and can set a default Market Segment, so the users have the Market Segment field in a new garment entry prefilled.

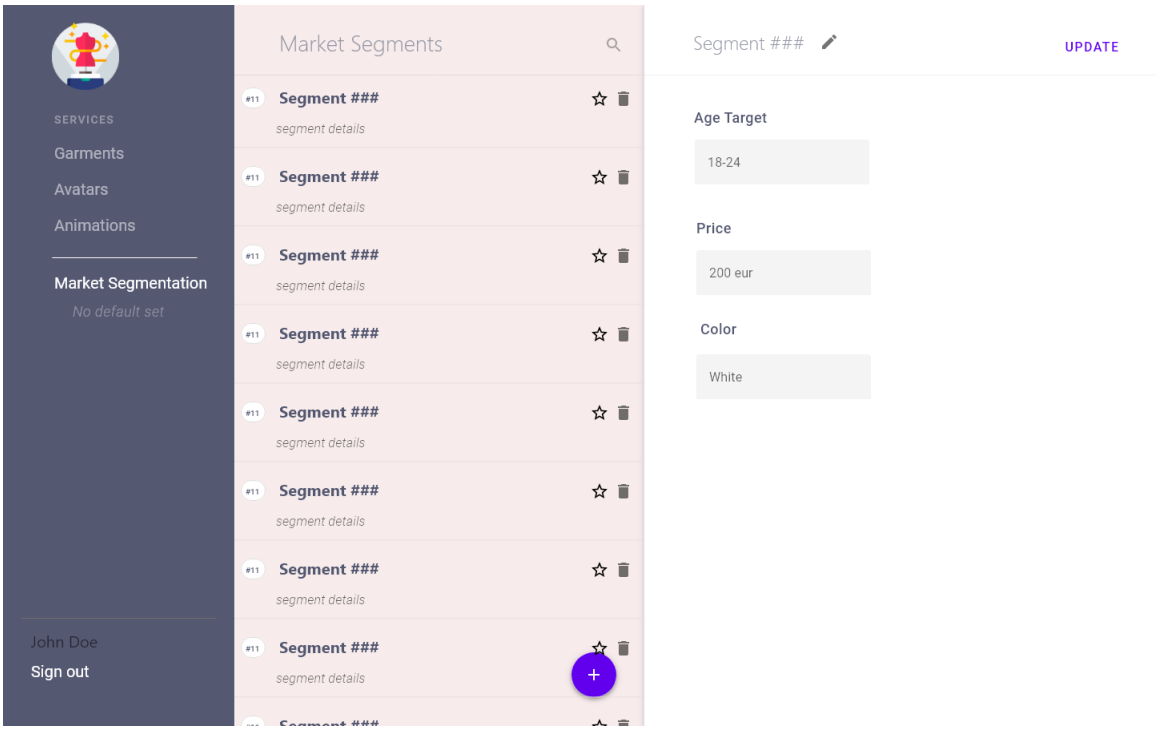

Figure 9: VR Designer Config app - Market Segmentation manager screen

3.3 Dress Me Up application

The Dress Me Up application is a web application adapted for mobile phones and desktop experiences alike targeting influencers, social media users and fashion lovers in general, to enhance their social media interactions. The function of this application is for the user to select a photo or video of themselves, along with a digital garment item that is provided within the application, send it to eTryOn's servers for simulating the digital garment on top of the user's image/video, visualize it inside the app after it has been rendered, and potentially share it on social media.

The Dress Me Up web application software is open source and available at https://gitlab.com/etryon/use-case-2/dress-me-up.

[2] https://etryon-h2020.eu/download/398

3.3.1 Implementation design update

There have been substantial changes to the design of the application while handling the initial implementation. The structure of the application has remained intact.

We used the 4th version of the Material-UI React framework[3], which offers a multitude of web components, instead of designing everything from scratch.

In **Figure 10** is the Sign In screen.

In **Figure 11** is the Account screen, where the users can now set their gender and size, in order for the system to load the appropriate garments when adding a new collection item. Users can also scan their body if they press the 'Scan Body' button.

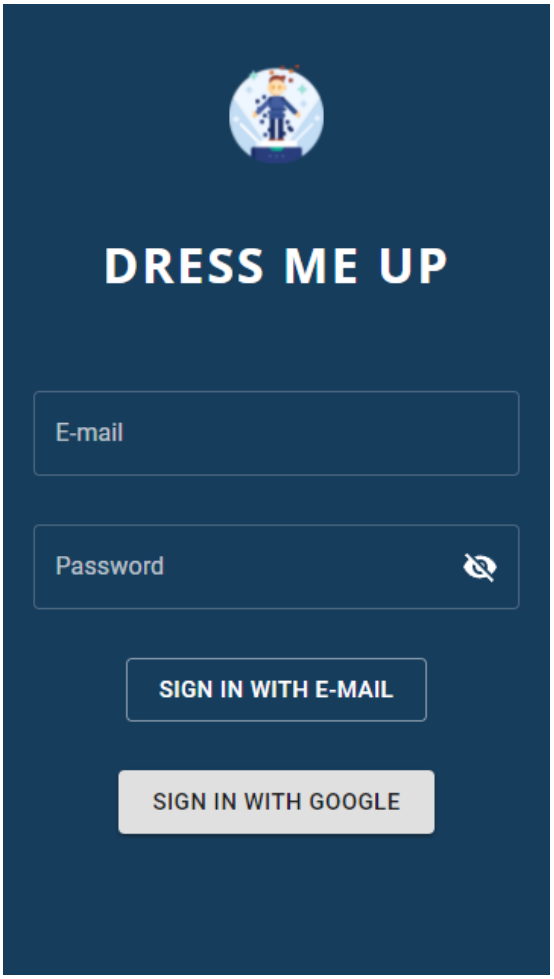

Figure 10: Dress Me Up - Sign In Screen

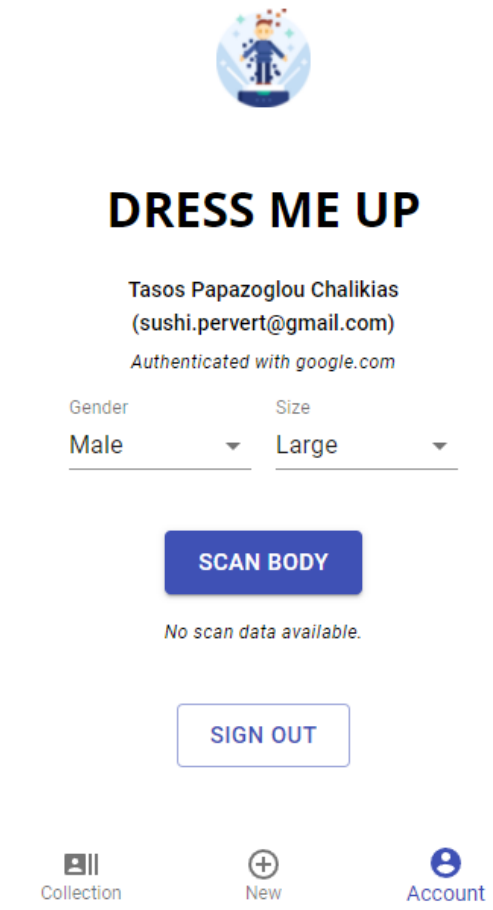

Figure 11: Dress Me Up - Account Screen

As seen in **Figure 12**, the user can upload a media file to the system in order for the media composition service to synthesize the new collection item. We have added the option to use the mobile or web camera in order to capture a photo or a video file, as well as the user to be able to select from an existing file. In **Figure 13** the view is populated with user selected photos.

[3] https://v4.mui.com/

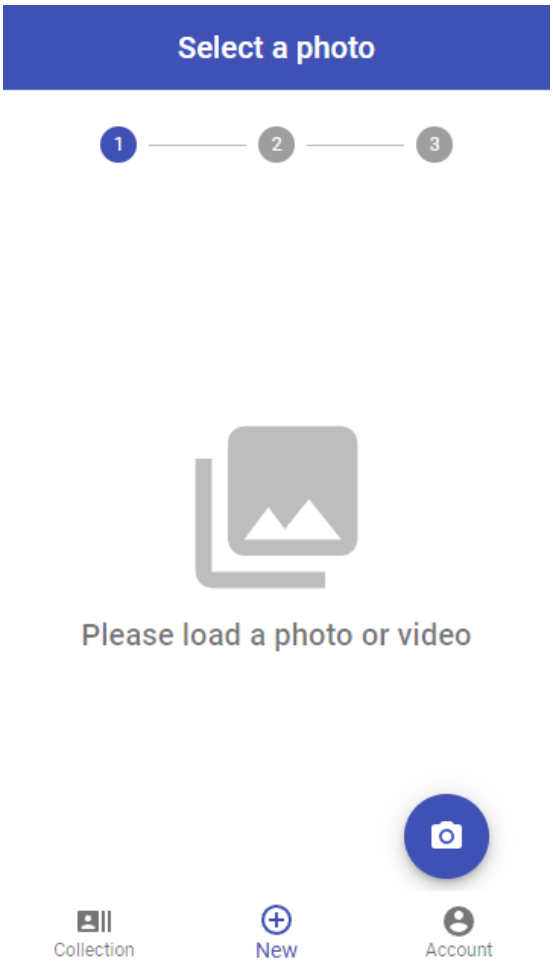

Figure 12: Dress Me Up - Add a new collection item (Step 1 - User media, empty)

Figure 13: Dress Me Up - Add a new collection item (Step 1 - User media, populated)

In **Figure 14** we can see that the garment list that is loaded is filtered based on the user preference on gender and size that was set in the Account view in **Figure 11**. The last step in the Add a new garment screen remains the same as before, as seen in **Figure 15**.

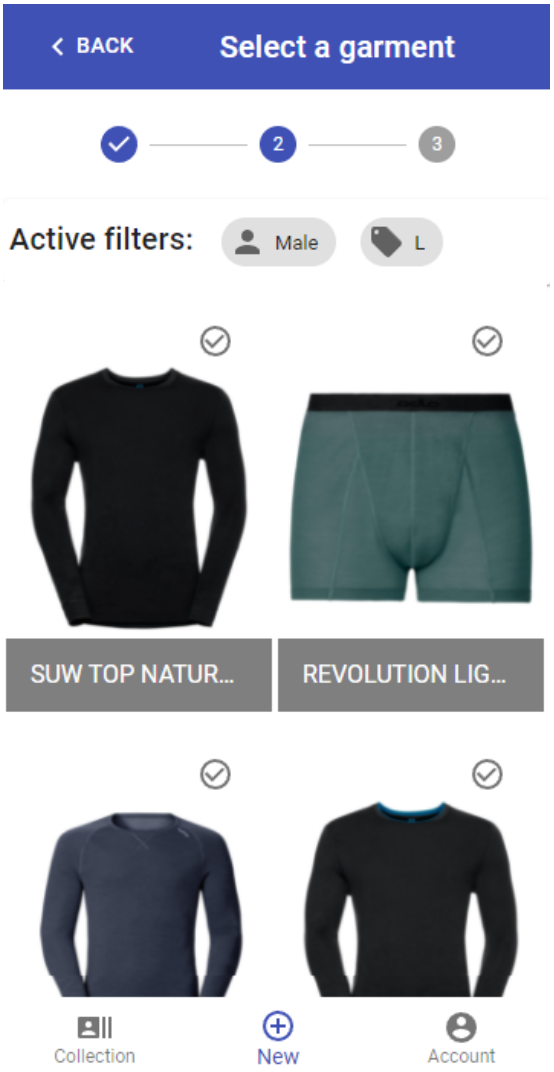

Figure 14: Dress Me Up - Add a new collection item (Step 2 - Garment list)

Figure 15: Dress Me Up - Add a new collection item (Step 3 - Upload preview)

The Collection view remains the same as before as seen in **Figures 16** & **17**.

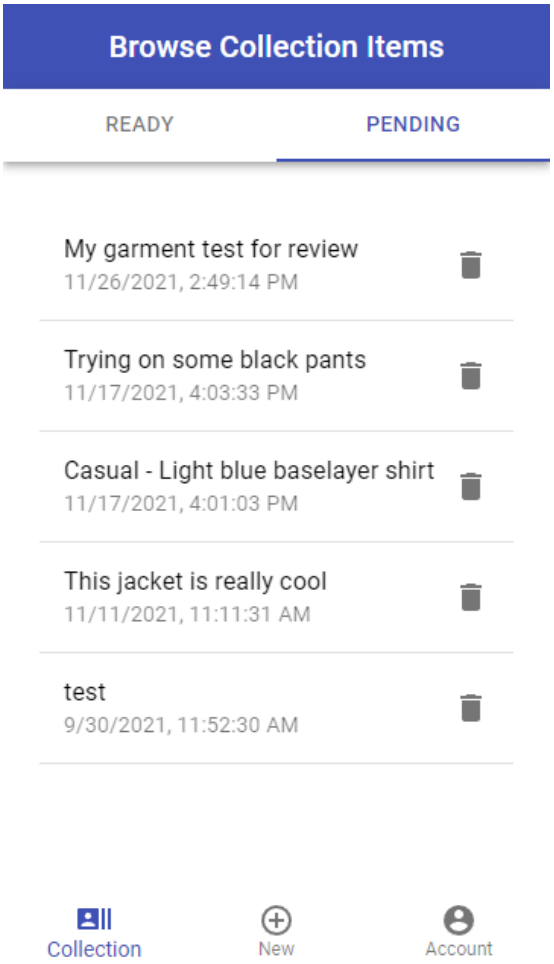

Figure 16: Dress Me Up - Collection (pending)

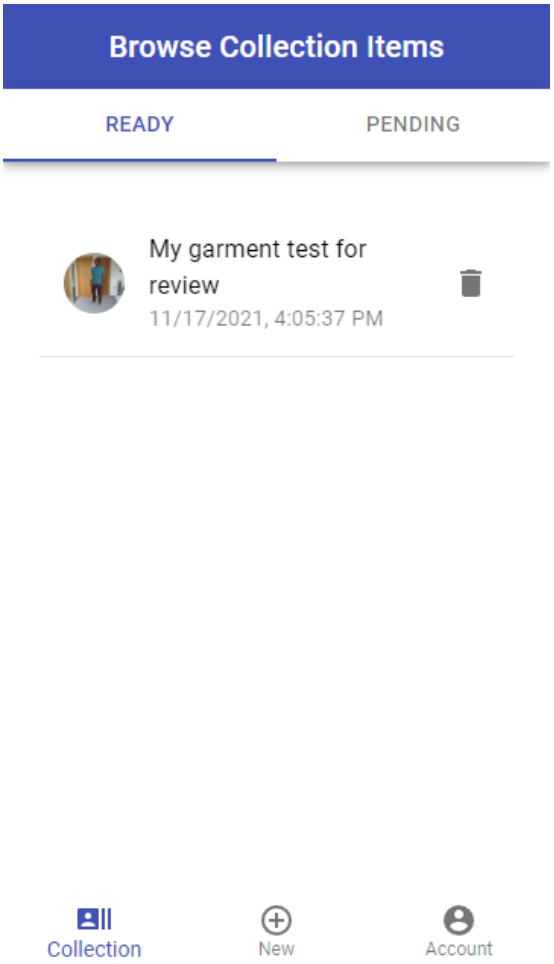

Figure 17: Dress Me Up - Collection (ready)

3.4 Magic Mirror application

The Magic Mirror mobile application is an AR dressing room for virtually trying on clothes at home with the capability to buy them through an interconnection to the ODLO e-shop[4]

The Magic Mirror mobile application software is open source and available at https://gitlab.com/etryon/use-case-3/MagicMirrorApp.

3.4.1 Primary User Interface

In order to speed up the testing and development process, we created a primary user interface that will be used for debugging purposes. We tested interscene operability by adding a splash screen, which leads the user to the AR view. Moreover, the AR view features 4 buttons:

1. **Show Garment**: Checks garment rendering and placing in space.
2. **Start Simulation**: Enables the Obi cloth simulation.
3. **Start Animation**: Showcases the natural movement of the garment affected by a walking animation.

[4] https://www.odlo.com/

4. **Start body tracking**: Renders the garment on top of a user that is tracked via the camera. This function currently does not support Obi cloth simulation.

Figures 18 & **19** demonstrate the look of the new user interface.

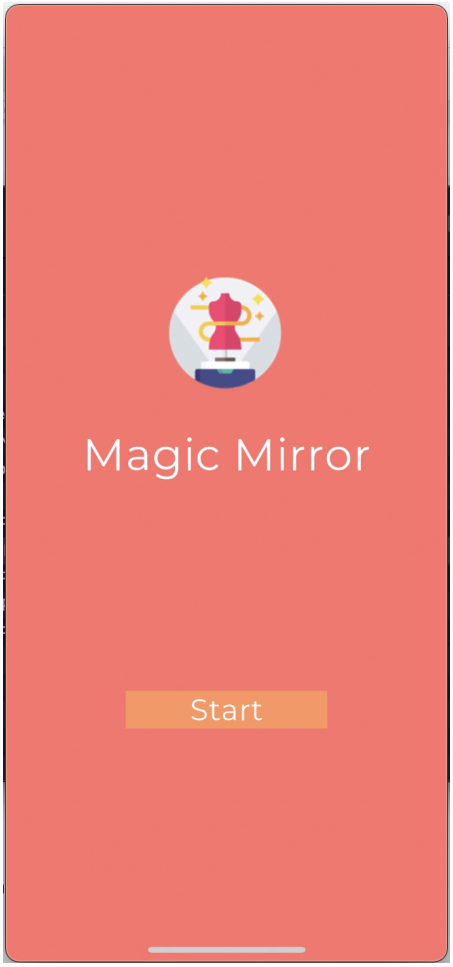

Figure 18: Magic Mirror - Splash screen

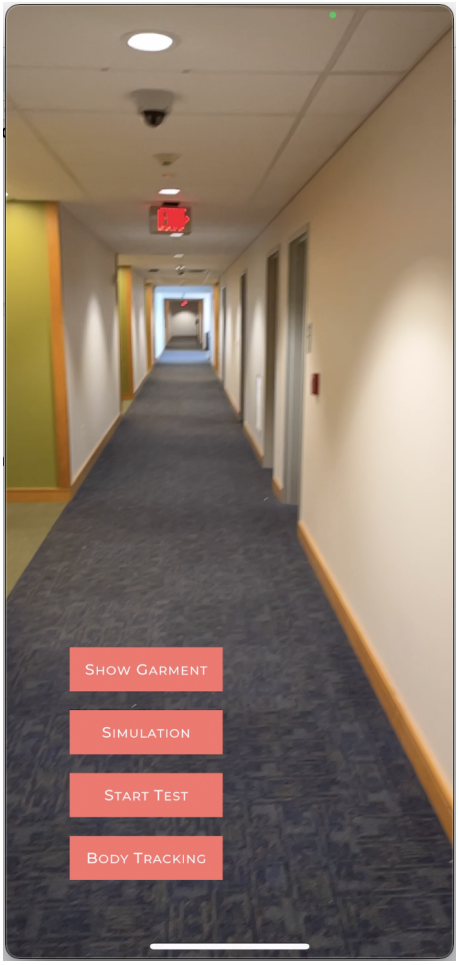

Figure 19: Magic Mirror - Simulation screen

3.4.2 Loading and Storing Asset Bundles

The human body tracking object in Unity, along with the garment, and the specified parameters will be stored as an asset bundle on the cloud and loaded at runtime by the application. This improves the efficiency of the application since there is no need to create these assets from scratch.

We have successfully developed the scripts needed for loading the asset bundles on the Unity scene. The creation of the asset bundles will happen on the cloud and this is a functionality that remains to be developed.

3.4.3 Creation of Obi Blueprints

We implemented the scripts to automatically create Obi blueprints for garments given the initial FBX file and the UV images containing the encoded Obi parameters. These parameters are used to specify the values of the skin radius, the backstop and the

backstop radius for every cloth particle. This is a function that will automatically run on the cloud every time a new garment is uploaded through the Admin console.

3.4.4 Garment Rigging according to ARKit specifications

In order to use the garments with ARKit's body tracking, their skeleton rig needs to abide by the specifications of ARKit. There has been successful development of automated scripts which produce the correct rig for the garment, ensuring the presence of the specified bone hierarchy, naming conventions and the correct orientation of the bones in the rig. In **Figure 20** the new rig according to the specifications by Apple's ARKit is shown.

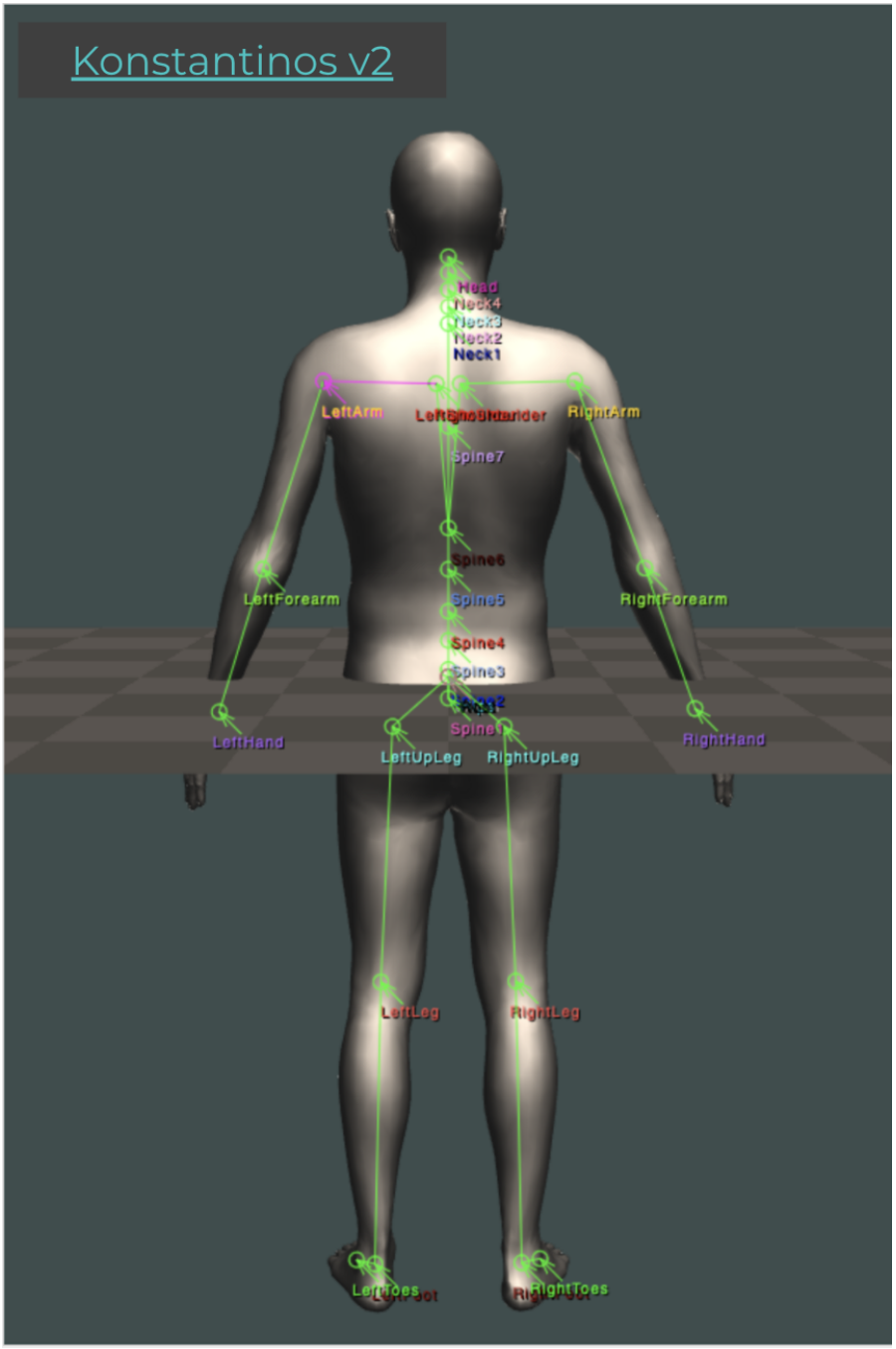

Figure 20: Magic Mirror - New skeleton rig according to Apple's ARKit specifications.

3.4.5 Stitching the garment parts

We have developed the functionality to stitch different garment parts together. This is important since the garments are exported by VStitcher in multiple pieces. In **Figure 21** we can see the stitched garment.

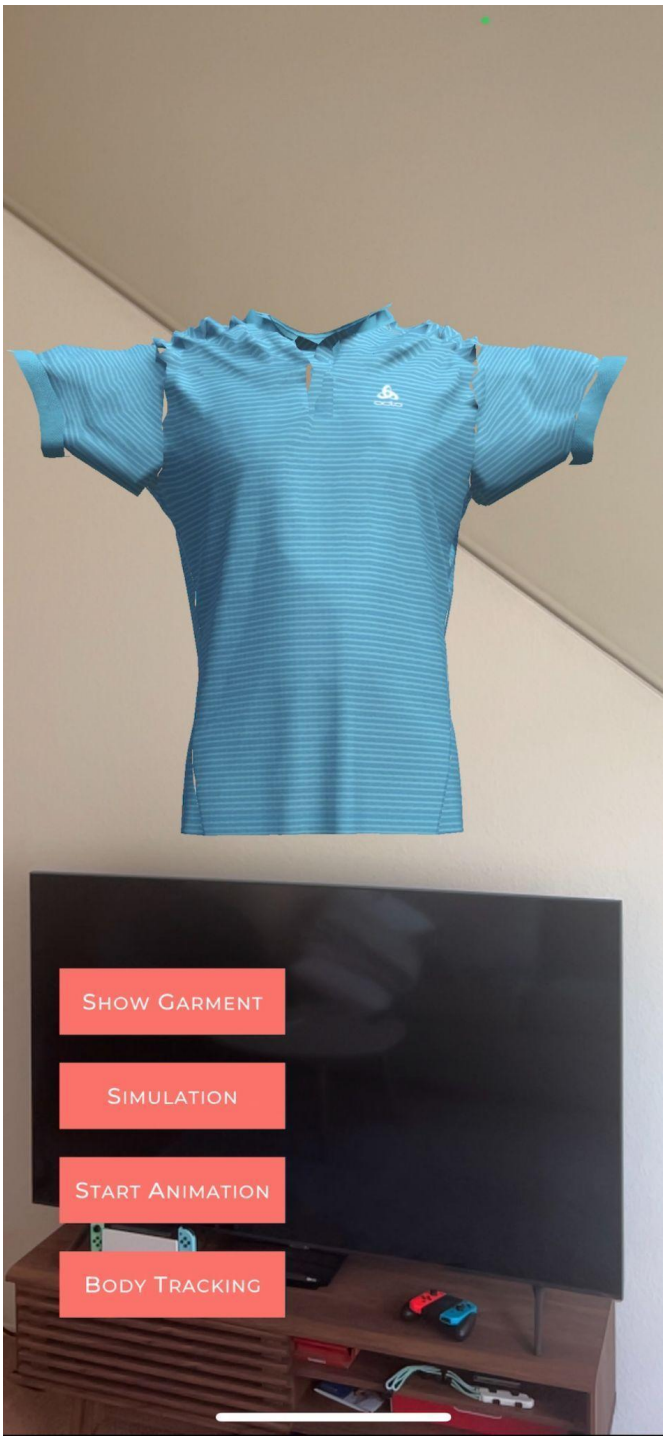

Figure 21: Magic Mirror - Stitched garment

Our stitching component presents some artifacts at the stitching points, which occur due to the reduction of triangles in the garment. The resolution of this issue is currently in progress.

3.5 Snapchat Lens filter

To increase visibility of the eTryOn project and the produced applications we considered the development of an AR experience as a Snapchat Lens effect.

We successfully integrated our Snapchat Lens with ODLO's promotional Scott biking suit, along with our previous work that featured a jumpsuit (**Figure 22**). The result can be seen in **Figure 23**. This Lens was published on the Snapchat platform and is available to use and share by the public using the Snapcode featured in **Figures 24** & **25** respectively.

The Snapchat Lens is now considered as a complete work and is available as open source at https://gitlab.com/etryon/etryon-snapchat-lens.

Figure 22: Snapchat Lens - Jumpsuit

Figure 23: Snapchat Lens - ODLO X SCOTT Biking suit

Figure 24: Snapchat Lens - Snapcode for Jumpsuit

Figure 25: Snapchat Lens - Snapcode for ODLO X SCOTT Biking suit

4. Conclusions & next actions

In this document we described the implementation updates for each use case. The implementation of eTryOns interaction systems is progressing smoothly, in order for the software to be ready to use in the upcoming pilots. Regarding the VR designer app, we have significantly improved the photo-realism of the VR space as well as the visualization realism of garment fabrics and textures, while we have beautified the UI and added more interactions based on the iterative testing with the designers. In the next period, besides further improving the UI and realism of the app, we will automate the process of adding new garments/avatars/animations through the connection to and the development of the back-office app. Regarding the DressMeUp, we have developed most of the UI functionalities and the application is connected with the project database. In the coming period, we will be adding more functionalities and integrating the eTryOn modules in the application. For the magic mirror app, we have developed the magic mirror view accompanied with a debugging interface. In the next period, we will develop the complete UI as it was designed in D5.1's mockups and we will integrate the additional modules of eTryOn to enable more functionalities (e.g. scanning). Finally, for the Snapchat Lens, our development work has been completed with the release of 2 filters. In the future, the process of making a new filter based on a new garment is automated and can be used upon request.

Table 1 below features all publicly available software repositories and there is software documentation available in the Appendix.

VR Designer	https://gitlab.com/etryon/uc1/vr-designer-app
Dress Me Up	https://gitlab.com/etryon/use-case-2/dress-me-up
Magic Mirror	https://gitlab.com/etryon/use-case-3/MagicMirrorApp
Snapchat Lens	https://gitlab.com/etryon/etryon-snapchat-lens

Table 1: Publicly available software repositories

5. Appendix - Software Documentation

5.1 VR Designer

5.1.1 Current Used Game Objects in Hierarchy

- **Light**: The general light setting affecting the scene.
- **Room and objects**: Some physical objects that make up all the room and the furniture.
- **Ceiling Lights**: Physical objects.
- **XR Interaction Manager**: A general VR component from the XR interaction toolkit.
- **XR Rig**: A game object that contains the VR camera and all the scripts regarding the VR controller mapping and all the custom scripts.
- **Reflection Probes**: Used for some better universal lighting.
- **Post Processing and Bloom effects**: Post processing effects and settings.
- **Spawner**: A game object containing the current alembic anima- tions in the scene.
- **Canvas (UI)**: It contains the UI buttons and interactions.

5.1.2 Scripts

5.1.2.1 XR-Toolkit auto scripts

- XR Interaction Manager
- XR Rig
- Input Action Manager

5.1.2.2 Custom scripts

5.1.2.2.1 Mover

This mover script allows the VR user to control the VR camera. Using the left controller Joystick the user can move the camera in the 3D space.

Attributes:

- (InputDevice) **device**: Store the VR device (left controller).
- (CharacterController) **character**: Controller of camera.
- (Vector2) **inputStick**: Vector2 of joystick input.
- (GameObject) **camera**: The camera.

Functions:

void Start(): Declare the VR left controller as the device and assign the camera as ”character” inside the scene so that we can move it with the controller.

void Update(): Left Joystick input and adjusts the cam- era to the corresponding input.

5.1.2.2.2 Rotate Object

This script allows the VR user to rotate the avatar in place using the right VR controller.

Attributes:

- (float) rotation**Speed** : Speed rotation multiplier.
- (InputDevice) **device** : Store the VR device (right controller).
- (Vector2) **controllerPos**: Stores the controller input in 2D.

Functions:

void Start(): Declare the VR right controller as the device.

void Update(): Wait for the right joystick input.

void Rotate(): Rotates the avatar based on the joystick input.

5.1.2.2.3 MnKRotate

Receives keyboard and mouse input and moves the VR camera accordingly. Initially used to enable easier movement inside the scene during the development process, since it does not require the VR headset controllers.

Attributes:

- (float) **rotationSpeed**: A multiplier for the speed rotation.
- (Vector3) **originalPosition**: Original position of camera.
- (Quaternion) **originalRotation**: Original rotation of camera.

Functions:

void Start(): Stores the initial state of the camera.

void Update(): Waits for WASD keyboard input or arrows input to move the camera accordingly.

5.1.2.2.4 Toggler(canvas)

A UI script to toggle the rest of the menu options on or off.

Attributes:

- (int) **click**: Flag to store the number of button clicks.
- (GameObject) **toggle**: The UI element to be toggled on and off.

Functions:

void clicks(): Spawns / despawns the UI elements.

5.1.2.2.5 Play

This script finds the current active avatar in the scene and dictates its Playing/Paused State.

Attributes:

- (GameObject) **Spawner**: A parent gameobject with all the available alembics as children.
- (GameObject) **cur-animation**: Currently activated animation.
- (int) **count**: Stores the number of children in Spawner.
- (int) **play**: Flag on the Play state.
- (GameObject) **clickPlay**: Changes color to the UI play button depending if the state is play or pause.

Functions:

void Start(): Finds the size of the available avatars.

void doPlay(): Resumes the animation if stopped.

void doPause(): Pauses the animation.

void PlayPause(): Finds the currently active avatar in the scene to checks the numbers on clicks on the Play/pause button.

void resetPause(): Resets the Playing/Paused state when- ever a new avatar spawns in the scene, so that everytime a new avatar is selected, its animation always plays automatically.

5.1.2.2.6 Avatar

Finds the currently active avatar in the scene and retrieves its PlayingDirector component. It also enables/disables the human avatar in the scene as to allow only.

Attributes:

- (GameObject) **spawner**: A parent gameobject with all the available alembics as children.
- (int) **count**: Stores the number of children in Spawner.
- (GameObject) **current**: Currently active animation.
- (int) **showav=0**: State of humanoid avatar to be on or off.
- (public) **GameObject Play**: The game avatar gameobject to be played or not.
- (float) **speed=10**: Animation speed multiplier.
- (PlayableDirector) **director**: Animator component.

Functions:

void Start(): Finds the number of possible active avatars and the UI Speed slider component.

void Update(): Always looks for the currently active avatar in the scene. The animation playing speed is updated every frame based on the slider input.

void showavatar(): Dictates the current state of showing the avatar or not.

void resetnext(): Resets the Avatar off state. Whenever a new avatar is selected the human body always shows.

void SpeedControl(): Function for the Speed Slider bar in the UI.

5.1.2.2.7 SpawnDespawn

A small script to enable/disable some UI elements.

Attributes:

- (int) **click**: Flag for the number of clicks on button.
- (GameObject) **spawn**: The UI objects to be toggled on/off.

Functions:

void Swap(): Spawns/despawns the corresponding UI.

5.1.2.2.8 ButtonSpawn

The main animation/avatar swapping script. Clicking on the corresponding animation buttons in the UI the script changes the currently playing garment/animation in the scene.

Attributes:

- (GameObject) **animation**: New animation to be displayed.
- (GameObject) **Spawner**: A parent gameobject with all the available alembics as children.
- (int) **count**: Stores the number of children in Spawner.

Functions:

void Start(): Finds the number of possible active avatars in the scene.

void AnimationSpawn(): Activates the specific animation and disables all the previously active ones.

5.2 Dress Me Up

5.2.1 Home

1. App.js

The main wrapper of the application.

Source: App.js, line 1

2. index.js

Entry point for the application.

Source: index.js, line 1

3. pages/Account.js

The Account page view

Source: pages/Account.js, line 1

4. pages/Collection.js

The Collection page view

Source: pages/Collection.js, line 1

5. pages/New.js

The Add a New Collection item page view

Source: pages/New.js, line 1

6. pages/Splash.js

The Splash page view

Source: pages/Splash.js, line 1

5.2.2 Modules

5.2.2.1 components/ImagePreview

Component for showing the image preview after the user has used the camera to capture a photo.

Parameters:

Name	Type	Description
`dataUri`	Object	Contains the photo details

Source: components/ImagePreview.js, line 4

Returns:

The image preview

Type: JSX.Element

5.2.2.2 components/NoCollectionDataSplash

This module creates a component - An empty view to show when no Collection data are available.

Parameters:

Name	Type	Description
`props`	Object	Contains a string that denotes in which state of the Collection view the user is in (Ready or Pending).

Source: components/NoCollectionDataSplash.js, line 7

Returns:

The 'No collection data' view

Type: JSX.Element

Examples

```
props.status = 'Ready'
```

```
props.status = 'Pending'
```

5.2.2.3 components/NoUserDataSplash

Module that invokes a component for when the user hasn't set up the user data.

Source: components/NoUserDataSplash.js, line 7

Returns:

The 'No User Data set up' view.

Type: JSX.Element

5.2.2.4 components/UnAuthedSplash

Module that invokes a component for when an unauthorized user tries to access a view of the application.

Source: components/UnAuthedSplash.js, line 7

Returns:

The 'Please login first' view.

Type: JSX.Element

5.2.3 Namespaces

5.2.3.1 Account

Account

Module for the Account view. Shows all the user information / settings.

Source: pages/Account.js, line 50

Methods

`(inner) handleGenderChange(event)`

Handles the change of the Gender dropdown component and uploads newly selected Gender string to Firebase

Parameters:

Name	Type	Description
`event`	Event	The dropdown selection event

Source: pages/Account.js, line 84

`(inner) handleSizeChange(event)`

Handles the change of the Size dropdown component and uploads newly selected Size string to Firebase

Parameters:

Name	Type	Description
`event`	Event	The dropdown selection event

Source: pages/Account.js, line 95

5.2.3.2 App

App

The main wrapper of the application

Source: App.js, line 57

Members

(static, constant) theme :Object

Saves an object of the created theme after a function is invoked to create the Material UI Theme.

Type: Object

Source: App.js, line 34

Methods

(static) ReturnLocationPath() → {string}

Returns the current url path the user is currently in.

Source: App.js, line 24

Returns:
location.pathname - url path.

Type: string

5.2.3.3 Collection

Collection

A component for showing the Collection View.

Source: pages/Collection.js, line 116

Classes

TabPanel

Methods

(static) a11yProps(index) → {Object}

Adds an aria-control value in each tab panel prop

Parameters:

Name	Type	Description
index	Number	The index id of the tab panel

Source: pages/Collection.js, line 69

Returns:

Type: Object

(static) CollectionListComponent(props) → {JSX.Element}

Component that loads all media files in a list

Parameters:

Name	Type	Description
props	Object	Object to denote in which view the user is in (Ready or Pending)

Source: pages/Collection.js, line 267

Returns:

Type: JSX.Element

(async, static) fetchItems(status) → {Promise.<void>}

Fetches collection items from Firebase

Parameters:

Name	Type	Description

status	String	The status of a collection item

Source: pages/Collection.js, line 310

Returns:

Returns

Type: Promise.<void>

Examples

```
status = 'Pending'
```

```
status = 'Ready'
```

(static) removeReadyItem(created)

It deletes a Ready collection item from Firebase.

Parameters:

Name	Type	Description
created	number	The created item timestamp

Source: pages/Collection.js, line 285

To Do:

Change the created param to a unique id instead of the collection item created timestamp

Add call to API in order for the deletion to work

(inner) downloadCollectionItem(dataUri)

Creates a link in order for the user to download the Collection Item media file.

Parameters:

Name	Type	Description
dataUri	URL	The Collection Item media file url link

Source: pages/Collection.js, line 158

(inner) handleChangeIndex(index)

Handles the event when tab changes to show the correct content.

Parameters:

Name	Type	Description
index	Number	The Tab element index id

Source: pages/Collection.js, line 142

(inner) handleChangeTab(event, newValue)

Handles the click event on a Tab element for navigation.

Parameters:

Name	Type	Description
event	Event	The click event
newValue	Number	The Tab element value

Source: pages/Collection.js, line 134

(inner) unloadCollectionItem()

Unloads a collection item url, in order to go back from previewing the file.

Source: pages/Collection.js, line 170

(inner) viewCollectionItem(item)

Sets Image source url in order to view a Collection Item.

Parameters:

Name	Type	Description
item	URL	Link URL that points to the collection item for preview

Source: pages/Collection.js, line 150

5.2.3.4 MobileCameraComponent

MobileCameraComponent

The Mobile / Web camera component that handles capturing a new photo.

Source: pages/New.js, line 851

Methods

(static) handleCameraBack()

Camera event handler for the 'back' event

Source: pages/New.js, line 903

(static) handleCameraError(error)

Camera capture error event handler

Parameters:

Name	Type	Description
error	Object	The error object

Source: pages/New.js, line 876

(static) handleCameraSaveImageToGrid()

Event handler that fires when the Image can be saved to the Grid.

Source: pages/New.js, line 912

(static) handleCameraStart(stream)

Camera capture start event handler

Parameters:

Name	Type	Description
stream		

Source: pages/New.js, line 885

(static) handleCameraStop(dataUri)

Camera capture stop event handler

Parameters:

Name	Type	Description
dataUri	String	

Source: pages/New.js, line 894

(static) handleSavePhoto()

Save Photo to local (download) event handler

Source: pages/New.js, line 921

(static) handleTakePhotoAnimationDone(dataUri)

Event Handler when the capture event fires

Parameters:

Name	Type	Description
dataUri	String	

Source: pages/New.js, line 866

5.2.3.5 NavBar

NavBar

The Navigation bar component.

Source: components/NavBar.js, line 28

Methods

(static) ReturnLocationPath() → {String}

Returns the current url path the user is in.

Source: components/NavBar.js, line 23

Returns:
location.pathname - url path.

Type: String

5.2.3.6 New

New

Component that creates the New collection item view.

Source: pages/New.js, line 169

Classes

TransitionUp

Members

`(static, constant) uniqueName :string`

Creates a Unique name for the new collection item (uid + filename + date timestamp in ms)

Type: string

Source: pages/New.js, line 353

`(inner, constant) fabActions :Array.<{icon: JSX.Element, name: String, event: Event}>`

The available Speed Dial actions that are shown when the component is open.

Type: Array.<{icon: JSX.Element, name: String, event: Event}>

Source: pages/New.js, line 270

Methods

`(static) getStepperStepContent(stepIndex) → {string}`

A function that returns the string that describes each upload step.

Parameters:

Name	Type	Description
stepIndex	Number	The current step index id.

Source: pages/New.js, line 156

Returns:

The current step string.

Type: string

`(static) getStepperSteps() → {Array.<string>}`

A function that returns an array of strings that feature all upload steps.

Source: pages/New.js, line 146

Returns:

An array of strings

Type: Array.<string>

`(async, inner) fetchGarments() → {Promise.<void>}`

Loads all garments from Firebase

Source: pages/New.js, line 450

To Do: Check paths when the app is updated according to spec.

Returns:

Type: Promise.<void>

`(inner) handleCloseBackdrop()`

Hides backdrop.

Source: pages/New.js, line 303

(inner) handleGarmentSelection(event)

Event Handler that fires when a new garment is selected and updates State.

Parameters:

Name	Type	Description
event	Event	Event when a garment is selected

Source: pages/New.js, line 217

(inner) handleMediaSelection(event)

Event Handler that fires when a new media file is selected and updates State.

Parameters:

Name	Type	Description
event	Event	Event when a media file is selected

Source: pages/New.js, line 207

(inner) handleSnackbarClose()

Hides Snackbar component.

Source: pages/New.js, line 314

(inner) handleSpeedDialClose()

Closes FAB Speed dial

Source: pages/New.js, line 249

(inner) handleSpeedDialOpen()

Opens FAB Speed dial

Source: pages/New.js, line 255

(inner) handleStepperBack()

Sets Active Step when a Previous Step event is fired.

Source: pages/New.js, line 235

(inner) handleStepperNext()

Sets Active Step when a Next Step event is fired.

Source: pages/New.js, line 229

(inner) handleStepperReset()

Sets Active Step when a Reset Step event is fired.

Source: pages/New.js, line 241

(async, inner) handleUpload(e) → {Promise.<void>}

Handles the upload of a new collection item.

Parameters:

Name	Type	Description
e	Event	Event that is fired when the submission of a new collection item starts.

Source: pages/New.js, line 334

To Do: Change media item selection by 'title' to a more concrete way.

Returns:

Type: Promise.<void>

(inner) onInputFileChange(event)

Event Handler that fires when a new image to upload is selected.

Parameters:

Name	Type	Description
event	Event	Event when a media file is loaded from a local file

Source: pages/New.js, line 193

To Do: This is only working for an image. Add use case for video

`(inner) openCameraOverlay()`

Renders the Mobile Camera Component

Source: pages/New.js, line 262

`(inner) openFileDialog()`

Opens File Dialog popup to select a media file from the filesystem.

Source: pages/New.js, line 184

`(inner) updateMediaGrid(media)`

Updates media grid with all available media files and updates state with selected media file

Parameters:

Name	Type	Description
`media`	string	Base64 encoded image string

Source: pages/New.js, line 280

5.2.3.7 Splash

Splash

Component for showing the login / signup screen of the app.

Source: pages/Splash.js, line 51

Members

`(inner, constant) location`

Get current location path url.

Source: pages/Splash.js, line 94

Methods

`(inner)` `handleClickShowPassword()`

Toggles the visibility of the typed password

Source: pages/Splash.js, line 79

`(inner)` `handleMouseDownPassword(event)`

Cancels the default mouse down event.

Parameters:

Name	Type	Description
`event`	Event	The mouse down event

Source: pages/Splash.js, line 87

Type Definitions

`handleChange`

Sets the password

Source: pages/Splash.js, line 72

5.2.4 Classes

5.2.4.1 CheckStyleRadio

CheckStyleRadio ( )

A custom Radio Button that uses Google Material Design icons.

Constructor

`new CheckStyleRadio()`

Source: components/CheckStyledRadio.js, line 18

Methods

`render() → {JSX.Element}`

Source: components/CheckStyledRadio.js, line 29

Returns:

The radio button component.

Type: JSX.Element

5.2.4.2 TabPanel

Collection.TabPanel() → {JSX.Element}

`new TabPanel() → {JSX.Element}`

Tab Panel Component

Source: pages/Collection.js, line 37

Returns:

The Tab Panel Component

Type: JSX.Element

5.2.4.3 IconButtonToggle

IconButtonToggle()

A Toggle Button component that features an Icon instead of a classic button.

Constructor

`new IconButtonToggle()`

Source: components/IconButtonToggle.js, line 10

Methods

`render() → {JSX.Element}`

Source: components/IconButtonToggle.js, line 22

Returns:
The Icon Toggle button component.

Type: JSX.Element

5.2.4.4 TransitionUp

New~TransitionUp(props) → {JSX.Element}

`new TransitionUp(props) → {JSX.Element}`

Sets the direction of the Speed Dial menu

Parameters:

Name	Type	Description
`props`	Object	

Source: pages/New.js, line 324

Returns:
Type: JSX.Element

5.2.5 Global

Members

`(constant) appTheme :Theme`

Main theme of the application

Type: Theme

Source: index.js, line 28

`(constant) firebaseChangeGender`

Function that updates the Gender property in Firebase

Source: helper.js, line 70

(constant) firebaseChangeSize

Function that updates the Size property in Firebase

Source: helper.js, line 40

Documentation generated by JSDoc 3.6.7.