

Rajiv Gandhi University Of Knowledge and Technologies Nuzvid

**(A.P. Government Act 18 of 2008)
RGUKT-NUZVID, Krishna Dist. - 521202
Tele Fax: 08656 – 235557/235150**



A Mini Project On “Virtual Try On Using Image Processing “

Submitted by

N181056	D.Gayathri
N180261	MD.Salma
N180263	Sk.Sameena
N180655	G.Krishnakumari
N181031	J.Gayathri



Department of Computer Science Engineering

2021-2022

CERTIFICATE OF COMPLETION

Certified that the mini-project work entitled “Virtual Try On Using Image Processing “is bonafied work carried out by

N181056

D.Gayathri

N180261

MD.Salma

N180263

Sk.Sameena

N180655

G.Krishnakumari

N181031

J.Gayathri

The report has been approved as it satisfies the academic requirements in respect of mini-project work prescribed for the course.

Mrs.Krishna Priya

Mini-Project Coordinator

Head of the Department

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

(A.P. Government Act 18 of 2008)

RGUKT-NUZVID, Krishna Dist - 521202

Tele Fax : 08656 – 235557/235150



CERTIFICATE OF EXAMINATION

This is to certify that the work entitled, " Virtual Try On Using Image Processing " is the Bonafide work of (N181056) D.Gayathri,(N180261) MD.Salma, (N180263) Sk.Sameena, (N180655) G.Krishnakumari, (N181031) J.Gayathri and here by accord our approval of it as a study carried out and presented in a manner required for its acceptance in pre final year of Bachelor of Technology for which it has been submitted. This approval does not necessarily endorse or accept every statement made an opinion expressed or conclusion drawn, as a recorded in this thesis. It only signifies the acceptance of this thesis for the purpose for which it has been submitted.

Mrs.K.Krishna Priya

Project supervisor

Assist. Prof Dept .of CSE
CSE

Examiner

Project supervisor

Lecturer Dept .of



RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

(A.P. Government Act 18 of 2008)
RGUKT-NUZVID, Krishna Dist. - 521202
Tele Fax: 08656 – 235557/235150

DECLARATION

We, hereby declare that the project report entitled as " Virtual Try On Using Image Processing " done by us under the guidance of Mrs Kona Krishna Priya is submitted for pre final year of Bachelor of Technology in Computer Science and Engineering at RGUKT - Nuzvid. We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Acknowledgment

We would like to express our deepest appreciation to all those who provided us the possibility to complete the project.

A special gratitude, we give to our mini project guidance Mrs K.Krishna Priya whose contribution in simulations, suggestions have encouraged us a lot.

N181056

D.Gayatri(N181056)

N180261

MD.Salma(N180261)

N180263

Sk.Sameena(N180263)

N181031

J.Gayatri(N181031)

N180655

G.Krishnakumari(N180655)

Virtual Try On Using Image Processing

CONTENTS

CERTIFICATES

ABSTRACT

1. INTRODUCTION

2. PROBLEM DEFINITION

Technologies Used

Hardware and Software Requirements

3. SYSTEM DESIGN

UML Diagrams

Use Case Diagram

Flow Diagrams

4. SAMPLE CODE

5. OUTPUT SCREENSHOTS

6. CONCLUSION AND REFERENCES

ABSTRACT

The virtual try-on project is a computer vision-based application that aims to revolutionize the retail industry by allowing customers to virtually try on accessories before making a purchase. Traditional shopping often poses challenges for customers, such as limited stock availability, sizing discrepancies, and the inability to physically try on products. Virtual try-on technology addresses these issues by leveraging computer vision algorithms to superimpose virtual accessories onto a user's image in real-time, creating an immersive and interactive shopping experience.

1.INTRODUCTION

Importance Of Virtual Try On Using Image Processing in Today's Era:

Virtual try-on project have completely transformed the way we approach shopping for glasses and accessories. Traditionally, customers had to rely on physical stores to try on different outfits, leading to time-consuming and sometimes frustrating experiences. However, with the introduction of virtual try-on project, customers can now enjoy the convenience of trying on items virtually, without leaving their homes.

By leveraging these technology , customers can visualize themselves wearing different products in real-time, providing them with a more immersive and interactive shopping experience.

One of the key advantages of virtual try-on projects is the enhanced customer experience they offer. Customers can now make more informed decisions by seeing how a particular glasses or accessories will look on them before making a purchase. This leads to increased customer satisfaction and reduced product returns, as customers have a clearer understanding of how the items will fit and suit them.

Additionally, virtual try-on projects have been proven to increase conversion rates. When customers can virtually try on clothes and see themselves wearing them, they are more likely to feel confident in their purchase decisions. This bridging of the gap between online and offline shopping experiences helps businesses drive sales and boosts their bottom line.

Another significant benefit is the level of personalization and customization virtual try-on projects provide. Customers can explore different styles, colors, and sizes tailored to their individual preferences and body types. This level of personalization enhances the overall shopping experience and helps customers find products that truly match their unique tastes.

Virtual try-on projects also contribute to cost and time savings for both customers and businesses. Customers no longer need to physically visit multiple stores to try on clothes, saving them time and transportation costs. Moreover, businesses can reduce the expenses associated with managing physical inventory, as virtual try-on allows them to showcase their entire product range digitally.

As technology continues to advance, This project expected to become even more sophisticated and widely adopted. They have already revolutionized the way we shop, offering enhanced customer experiences, increased conversion rates, personalization, cost and time savings, trend exploration, and sustainability. These projects are reshaping the fashion and retail industries and are poised to play a pivotal role in the future of shopping.

EXISTING SYSTEM:

The existing system is basically a Web based system. In that we can't try on Necklaces but only glasses.

PROPOSED SYSTEM:

In our System along with the Spectacles we have added extra functionality of trying necklaces.

This Was a Desktop Application With the attracting User Interface

2 .Problem Definition

Technologies used:

IDE: PyCharm

PyCharm is an Integrated Development Environment (IDE) specifically designed for Python development. It is developed by JetBrains, a renowned software development company. PyCharm provides a wide range of features and tools to facilitate efficient and productive Python programming.

PyCharm is available in two editions: PyCharm Community Edition (free and open-source) and PyCharm Professional Edition (commercial with additional advanced features).

Overall, PyCharm is a comprehensive and feature-rich IDE that empowers Python developers with a seamless development experience, boosting productivity and code quality.

Programming Language: Python

Python's versatility and rich ecosystem make it suitable for a wide range of applications, including web development, data analysis, scientific computing, artificial intelligence, machine learning, automation, and more. Its simplicity, readability, and extensive libraries have made Python a preferred choice for both beginners and experienced developers alike.

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991. Python has gained immense popularity in various domains, including web development, data analysis, machine learning, scientific computing, and automation.

Open cv :

OpenCV (Open Source Computer Vision) is an open-source computer vision and machine learning library. It provides a comprehensive set of functions and algorithms for image and video processing, object detection and tracking, feature extraction, and more. OpenCV is widely used in various domains, including robotics, augmented reality, healthcare, surveillance, and automotive.

OpenCV is primarily written in C++, but it also provides APIs for Python, Java, and other programming languages. This allows developers to use OpenCV with their preferred programming languages and environments. Overall, OpenCV is a powerful and versatile library for computer vision tasks, offering a wide range of capabilities to developers and researchers in the field.

Dlib(Face_Recognition):

Dlib is a popular open-source library for computer vision, machine learning, and image processing. It provides a comprehensive set of tools and algorithms for various tasks, including face detection, facial landmark detection, face recognition, object detection and tracking, image segmentation, and more.

dlib includes highly accurate face detection algorithms, including a frontal face detector (get_frontal_face_detector())

dlib provides a powerful facial landmark detection algorithm (`shape_predictor()`) that can localize key points on a face, such as eyes, nose, mouth, and jawline. This allows for various applications, including face alignment, expression analysis, and facial feature-based recognition.

Python Framework: (PyQt5):

PyQt5 is a Python binding for the Qt framework, which is a popular and powerful cross-platform application development framework. PyQt5 allows developers to create graphical user interfaces (GUIs) for desktop applications using Python.

PyQt5 offers extensive documentation, tutorials, and community support, making it a popular choice for developing desktop applications with Python and Qt. It provides a robust and flexible framework for creating modern and user-friendly graphical interfaces for a wide range of use cases.

PyQt5 enables the development of cross-platform applications that can run on different operating systems, such as Windows, macOS, and Linux, without significant modifications. This is because it is built on the Qt framework, which provides a consistent API and functionality across platforms.

Software Requirements

- Coding Language : Python
- Operating system microprocessor : Windows operating system (64-bit) with 8 gb
- Tools: PyCharm,Idle Python,Visual Studio Code

Hardware Requirements

- System Processor : Intel(R) Core(TM) i5-3320M CPU @2.60GHz
- Hard Disk : 512 GB SSD
- System Type : 64-bit Operating system,x64-based Processor
- Ram : 16 GB

3.SYSTEM DESIGN

UML DIAGRAMS:

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

These methods were already evolving toward each other independently. It made sense to continue that evolution together rather than apart, eliminating the potential for any unnecessary and gratuitous differences that would further confuse users.

By unifying the semantics and notation, they could bring some stability to the object-oriented marketplace, allowing projects to settle on one mature modeling language and letting tool builders focus on delivering more useful features.

They expected that their collaboration would yield improvements in all three earlier methods, helping them to capture lessons learned and to address problems that none of their methods previously handled well

Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.

Provide extensibility and specialization mechanisms to extend the core concepts.

Be independent of particular programming languages and development processes.

Provide a formal basis for understanding the modeling language. Encourage the

growth of the OO tools market.

Support higher-level development concepts such as collaborations, frameworks, patterns and components.

Integrate best practices. UML has three building blocks: Things - The basic building elements of a model; Relationships - How the things are tied together; Diagrams -

Graphical representation of element collection

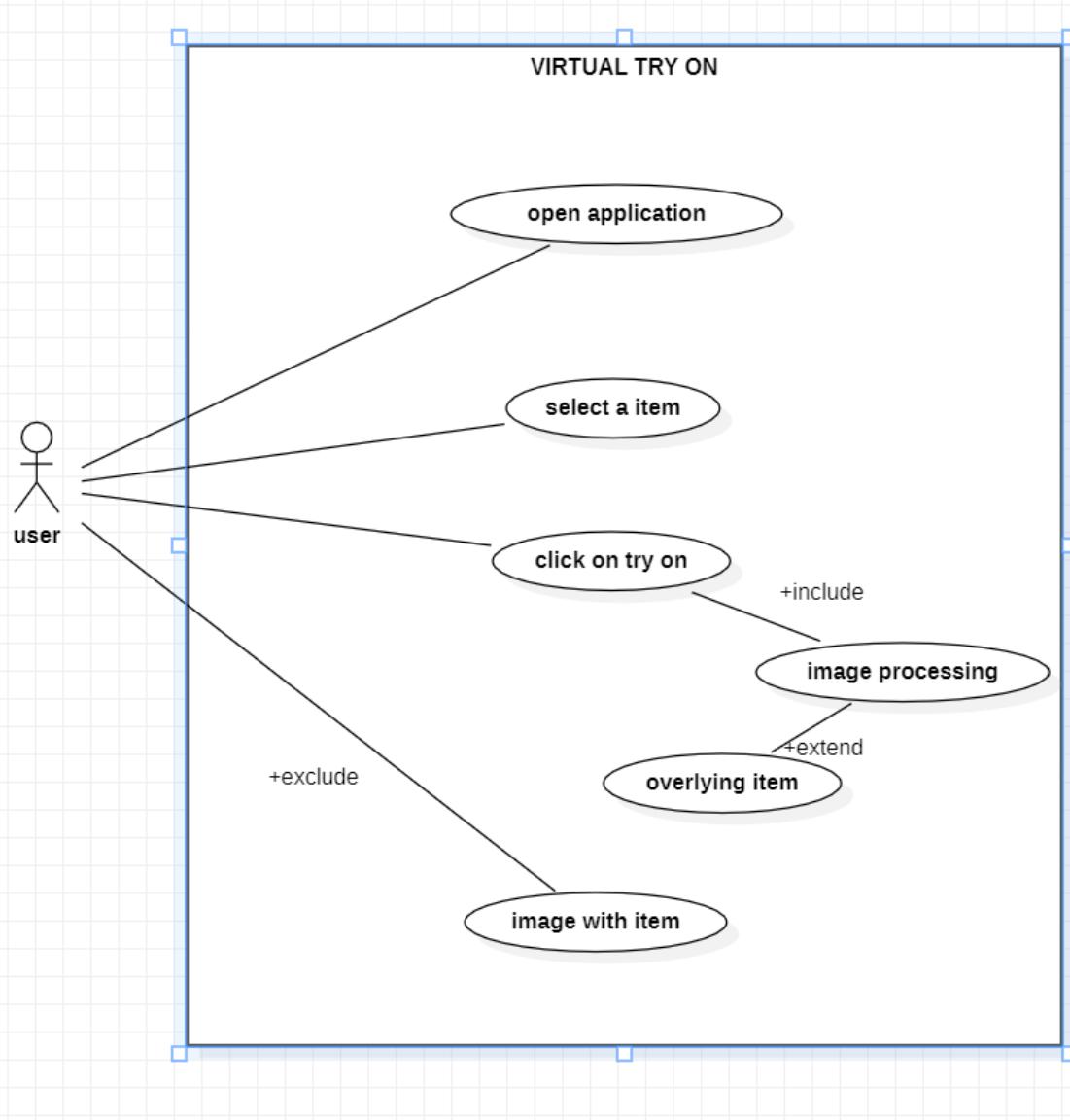
Diagramming techniques

Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tools contain the following

Analysis phase

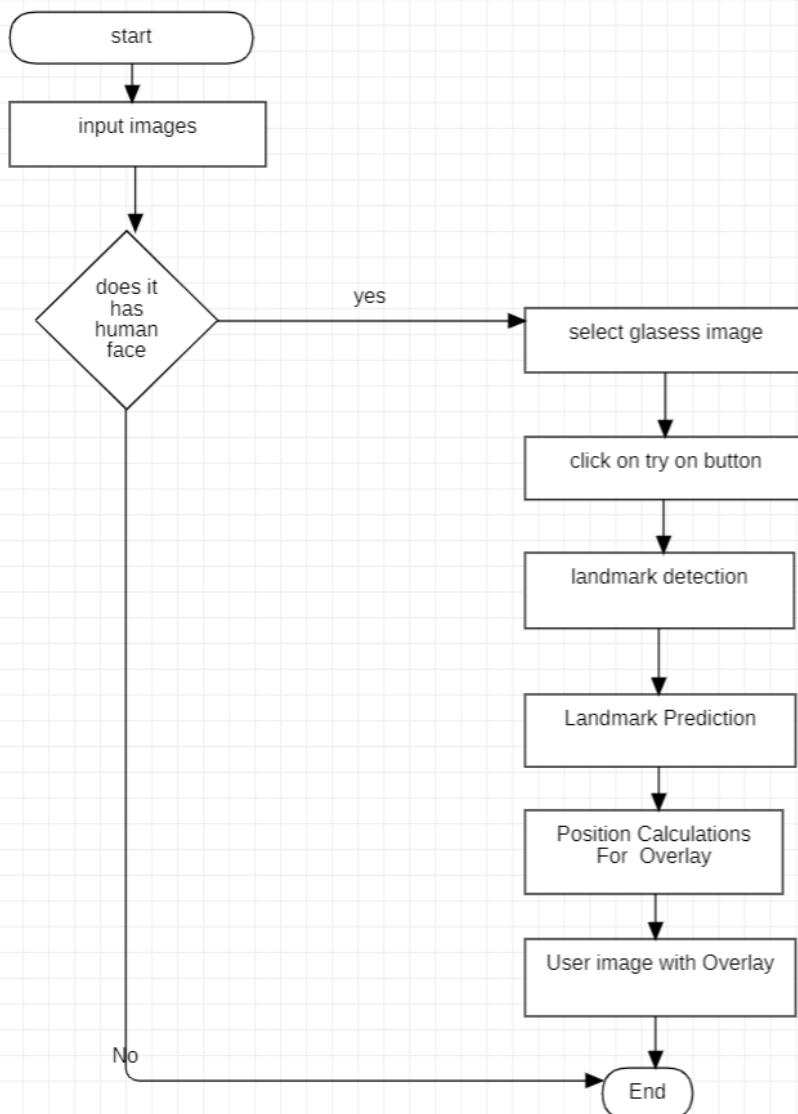
Use Case Diagram:

Use case diagrams help to capture the functional requirements of a system and illustrate the interactions between actors (users or external entities) and the system itself. They provide a high-level view of the system's behaviour and the functionalities it offers.



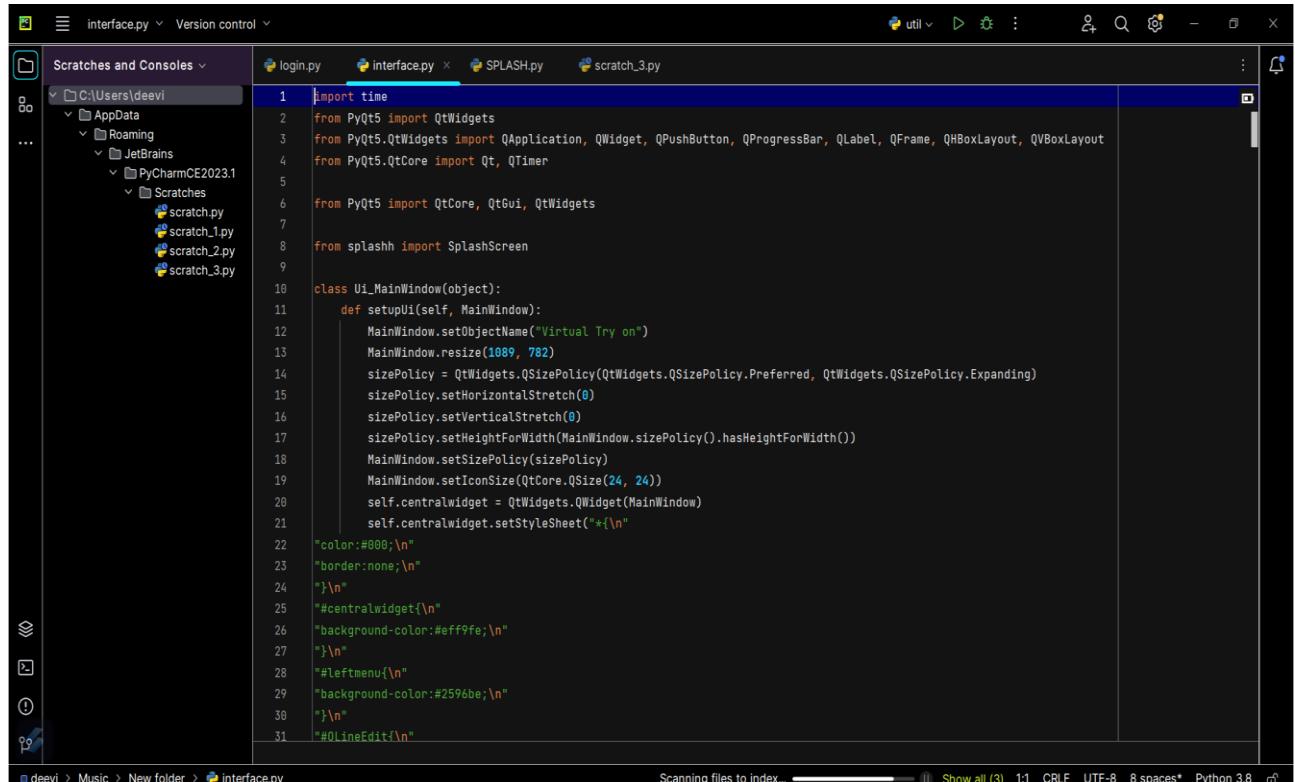
FLOW DIAGRAM

A flowchart is a picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan.



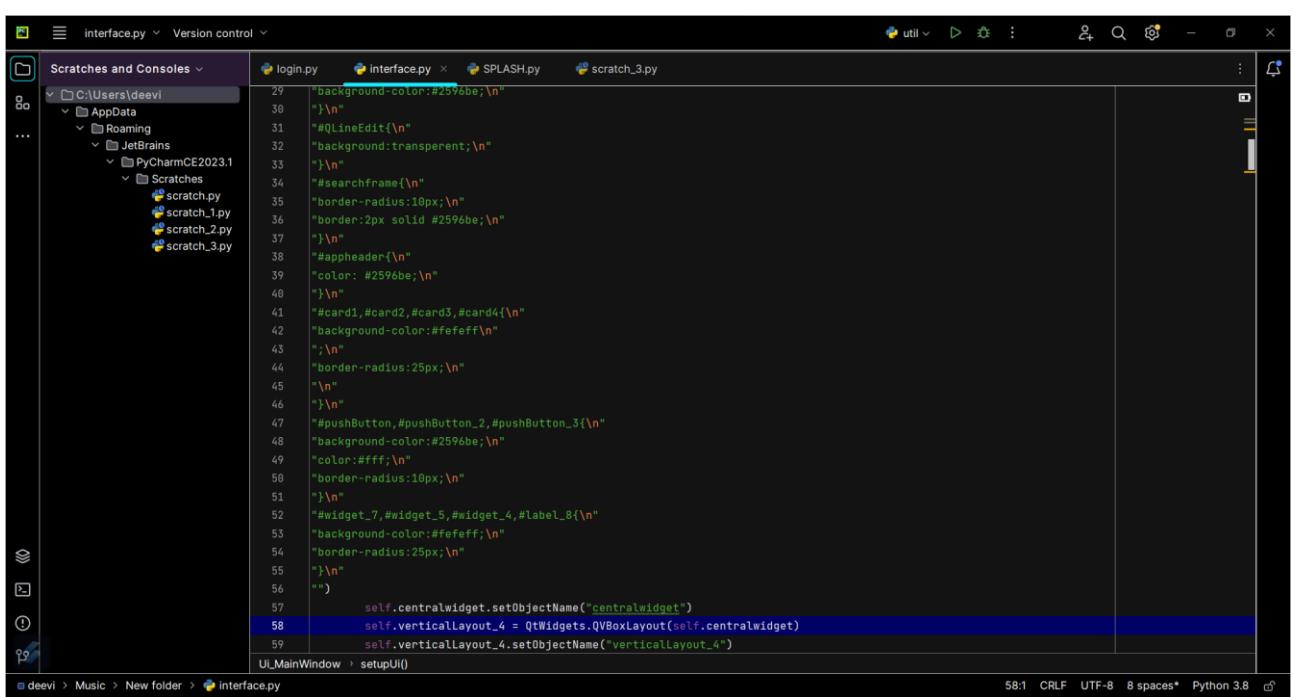
4.SAMPLE CODE

@User Interface



```
import time
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QProgressBar, QLabel, QFrame, QVBoxLayout, QHBoxLayout
from PyQt5.QtCore import Qt, QTimer
from PyQt5 import QtCore, QtGui, QtWidgets
from splash import SplashScreen

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("Virtual Try on")
        MainWindow.resize(1089, 782)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setIconSize(QtCore.QSize(24, 24))
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setStyleSheet("*{\n    color:#000;\n    border:none;\n}\n\n#centralwidget{\n    background-color:#eff9fe;\n}\n\n#leftmenu{\n    background-color:#2596be;\n}\n\n#LineEdit{\n    background-color:#2596be;\n    border-radius:10px;\n    border:2px solid #2596be;\n}\n\n#appheader{\n    color: #2596be;\n}\n\n#card1,#card2,#card3,#card4{\n    background-color:#fefeff;\n    border-radius:25px;\n}\n\nQPushButton,#pushButton_2,#pushButton_3{\n    background-color:#2596be;\n    color:#fff;\n    border-radius:10px;\n}\n\n#widget_7,#widget_5,#widget_4,#label_8{\n    background-color:#fefeff;\n    border-radius:25px;\n}\n\nself.centralwidget.setObjectName("centralwidget")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout_4.setObjectName("verticalLayout_4")
```



```
background-color:#2596be;\n}\n\n#LineEdit{\n    background:transparent;\n}\n\n#searchframe{\n    border-radius:10px;\n    border:2px solid #2596be;\n}\n\n#appheader{\n    color: #2596be;\n}\n\n#card1,#card2,#card3,#card4{\n    background-color:#fefeff;\n    border-radius:25px;\n}\n\nQPushButton,#pushButton_2,#pushButton_3{\n    background-color:#2596be;\n    color:#fff;\n    border-radius:10px;\n}\n\n#widget_7,#widget_5,#widget_4,#label_8{\n    background-color:#fefeff;\n    border-radius:25px;\n}\n\nself.centralwidget.setObjectName("centralwidget")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout_4.setObjectName("verticalLayout_4")
```

```
self.centralwidget.setObjectName("centralwidget")
self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.centralwidget)
self.verticalLayout_4.setObjectName("verticalLayout_4")
self.mainbody = QtWidgets.QWidget(self.centralwidget)
self.mainbody.setAutoFillBackground(False)
self.mainbody.setObjectName("mainbody")
self.verticalLayout_4.addWidget(self.mainbody)
self.headerFrame = QtWidgets.QWidget(self.mainbody)
self.headerFrame.setAutoFillBackground(False)
self.headerFrame.setObjectName("headerFrame")
self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.headerFrame)
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
self.widget = QtWidgets.QWidget(self.headerFrame)
self.widget.setObjectName("widget")
self.horizontalLayout_2.addWidget(self.widget)
self.widget.setLayout(QtWidgets.QHBoxLayout(self.widget))
self.widget.layout().setObjectName("horizontalLayout_3")
self.menuBtn = QtWidgets.QPushButton(self.widget)
self.menuBtn.setObjectName("menuBtn")
self.menuBtn.setText("")
icon = QtGui.QIcon()
icon.addPixmap(QtGui.QPixmap(":/blueIcons/icons/blue/menu.svg"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
self.menuBtn.setIcon(icon)
self.menuBtn.setIconSize(QtCore.QSize(24, 24))
self.menuBtn.setObjectName("menuBtn")
self.horizontalLayout_3.addWidget(self.menuBtn, 0, QtCore.Qt.AlignTop)
self.appheader = QtWidgets.QLabel(self.widget)
font = QtGui.QFont()
font.setPointSize(15)
font.setBold(True)
font.setWeight(75)
```

```
self.appheader.setFont(font)
self.appheader.setObjectName("appheader")
self.horizontalLayout_3.addWidget(self.appheader)
self.widget_2 = QtWidgets.QWidget(self.headerFrame)
self.widget_2.setObjectName("widget_2")
self.horizontalLayout_2.addWidget(self.widget_2, 0, QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)
self.widget_2.setLayout(QtWidgets.QHBoxLayout(self.widget_2))
self.horizontalLayout_2.setContentsMargins(0, 0, 0)
self.horizontalLayout_6.setObjectname("horizontalLayout_6")
self.accountbtn = QtWidgets.QPushButton(self.widget_2)
self.accountbtn.setMinimumSize(QtCore.QSize(24, 24))
self.accountbtn.setMaximumSize(QtCore.QSize(30, 30))
self.accountbtn.setText("")
icon1 = QtGui.QIcon()
icon1.addPixmap(QtGui.QPixmap(":/blueIcons/icons/blue/user.svg"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
self.accountbtn.setIcon(icon1)
self.accountbtn.setIconSize(QtCore.QSize(32, 32))
self.accountbtn.setObjectName("accountbtn")
self.horizontalLayout_6.addWidget(self.accountbtn, 0, QtCore.Qt.AlignTop)
self.horizontalLayout_2.addWidget(self.widget_2, 0, QtCore.Qt.AlignRight|QtCore.Qt.AlignTop)
self.verticalLayout.addWidget(self.headerFrame, 0, QtCore.Qt.AlignTop)
self.cardsFrame = QtWidgets.QWidget(self.mainbody)
self.cardsFrame.setObjectName("cardsFrame")
self.horizontalLayout_7 = QtWidgets.QHBoxLayout(self.cardsFrame)
self.horizontalLayout_7.setContentsMargins(0, 0, 0)
self.horizontalLayout_7.setSpacing(20)
self.horizontalLayout_7.setObjectName("horizontalLayout_7")
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays a file tree under 'Scratches and Consoles' with several Python files: login.py, interface.py (the active file), SPLASH.py, and scratch_3.py. The code editor window contains the following Python code:

```
111 self.horizontalLayout_7.setContentsMargins(0, 0, 0, 0)
112 self.horizontalLayout_7.setSpacing(20)
113 self.horizontalLayout_7.setObjectName("horizontalLayout_7")
114 self.card2 = QtWidgets.QFrame(self.cardsFrame)
115 self.card2.setMinimumSize(QtCore.QSize(160, 0))
116 self.card2.setMaximumSize(QtCore.QSize(16777215, 150))
117 self.card2 setFrameShape(QtWidgets.QFrame.StyledPanel)
118 self.card2 setFrameShadow(QtWidgets.QFrame.Raised)
119 self.card2.setObjectName("card2")
120 self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.card2)
121 self.verticalLayout_2.setContentsMargins(0, -1, -1, -1)
122 self.verticalLayout_2.setObjectName("verticalLayout_2")
123 self.home = QtWidgets.QFrame(self.frame)
124 self.home setFrameShape(QtWidgets.QFrame.StyledPanel)
125 self.home setFrameShadow(QtWidgets.QFrame.Raised)
126 self.home.setObjectName("home")
127 self.horizontalLayout_8 = QtWidgets.QHBoxLayout(self.home)
128 self.horizontalLayout_8.setObjectName("horizontalLayout_8")
129 self.label = QtWidgets.QLabel(self.home)
130 font = QtGui.QFont()
131 font.setPointSize(14)
132 font.setBold(True)
133 font.setItalic(False)
134 font.setWeight(75)
135 self.label.setFont(font)
136 self.label.setObjectName("label")
137 self.horizontalLayout_8.addWidget(self.label)
138 self.label_3 = QtWidgets.QLabel(self.home)
139 self.label_3.setMinimumSize(QtCore.QSize(40, 40))
140 self.label_3.setMaximumSize(QtCore.QSize(40, 40))
141 self.label_3.setText("")
```

The status bar at the bottom indicates the file path as 'deevi > Music > New folder > interface.py', and the current time as '14:07'. The encoding is set to 'CRLF' and the font size to '8 spaces* Python 3.8'.

This screenshot is nearly identical to the one above, showing the same PyCharm interface and code editor. The code editor window contains the same Python code as the first screenshot, with the cursor positioned at line 140. The status bar at the bottom indicates the file path as 'deevi > Music > New folder > interface.py', and the current time as '16:07'. The encoding is set to 'CRLF' and the font size to '8 spaces* Python 3.8'.

Screenshot of PyCharm IDE showing the file `interface.py`. The code defines a `Ui_MainWindow` class with a `setupUi()` method. The code sets up various UI components including labels, layouts, and frames, using Qt's `QLabel`, `QFrame`, and `QLayout` classes. It also uses `QtGui.QPixmap` for icons and `QtCore.QSize` for sizes.

```
167     font.setItalic(False)
168     font.setWeight(75)
169     self.label_5.setFont(font)
170     self.label_5.setObjectName("label_5")
171     self.horizontalLayout_9.addWidget(self.label_5, 0, QtCore.Qt.AlignLeft)
172     self.label_6 = QtWidgets.QLabel(self.tryon)
173     self.label_6.setMinimumSize(QtCore.QSize(40, 40))
174     self.label_6.setMaximumSize(QtCore.QSize(40, 40))
175     self.label_6.setText("")
176     self.label_6.setPixmap(QtGui.QPixmap(":/blueIcons/icons/blue/aperture.svg"))
177     self.label_6.setScaledContents(True)
178     self.label_6.setObjectName("label_6")
179     self.horizontalLayout_9.addWidget(self.label_6)
180     self.verticalLayout_3.addWidget(self.tryon, 0, QtCore.Qt.AlignTop)
181     self.horizontalLayout_7.addWidget(self.card1)
182     self.card4 = QtWidgets.QFrame(self.cardsFrame)
183     self.card4.setMinimumSize(QtCore.QSize(160, 0))
184     self.card4.setFrameShape(QtWidgets.QFrame.StyledPanel)
185     self.card4.setFrameShadow(QtWidgets.QFrame.Raised)
186     self.card4.setObjectName("card4")
187     self.verticalLayout_5 = QtWidgets.QVBoxLayout(self.card4)
188     self.verticalLayout_5.setObjectName("verticalLayout_5")
189     self.contactus = QtWidgets.QFrame(self.card4)
190     self.contactus.setFrameShape(QtWidgets.QFrame.StyledPanel)
191     self.contactus.setFrameShadow(QtWidgets.QFrame.Raised)
192     self.contactus.setObjectName("contactus")
193     self.horizontalLayout_11 = QtWidgets.QHBoxLayout(self.contactus)
194     self.horizontalLayout_11.setObjectName("horizontalLayout_11")
195     self.label_12 = QtWidgets.QLabel(self.contactus)
196     self.label_12.setMinimumSize(QtCore.QSize(40, 40))
197     self.label_12.setMaximumSize(QtCore.QSize(46, 40))
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
Ui_MainWindow > setupUi()
```

Screenshot of PyCharm IDE showing the file `interface.py`. The code defines a `Ui_MainWindow` class with a `setupUi()` method. The code sets up various UI components including labels, layouts, and frames, using Qt's `QLabel`, `QFrame`, and `QLayout` classes. It also uses `QtGui.QPixmap` for icons and `QtCore.QSize` for sizes. This version of the code includes additional styling and layout configurations compared to the first screenshot.

```
194     self.horizontalLayout_11.setObjectName("horizontalLayout_11")
195     self.label_12 = QtWidgets.QLabel(self.contactus)
196     self.label_12.setMinimumSize(QtCore.QSize(40, 40))
197     self.label_12.setMaximumSize(QtCore.QSize(46, 40))
198     self.label_12.setText("")
199     self.label_12.setPixmap(QtGui.QPixmap(":/blueIcons/icons/blue/phone-call.svg"))
200     self.label_12.setScaledContents(True)
201     self.label_12.setObjectName("label_12")
202     self.horizontalLayout_11.addWidget(self.label_12)
203     self.label_11 = QtWidgets.QLabel(self.contactus)
204     font = QtGui.QFont()
205     font.setPointSize(14)
206     font.setBold(True)
207     font.setItalic(False)
208     font.setWeight(75)
209     self.label_11.setFont(font)
210     self.label_11.setObjectName("label_11")
211     self.horizontalLayout_11.addWidget(self.label_11)
212     self.verticalLayout_5.addWidget(self.contactus)
213     self.horizontalLayout_7.addWidget(self.card4)
214     self.verticalLayout.addWidget(self.cardsFrame, 0, QtCore.Qt.AlignTop)
215     self.verticalLayout_4.addWidget(self.mainbody, 0, QtCore.Qt.AlignTop)
216     self.widget_3 = QtWidgets.QWidget(self.centralwidget)
217     sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
218     sizePolicy.setHorizontalStretch(0)
219     sizePolicy.setVerticalStretch(0)
220     sizePolicy.setHeightForWidth(self.widget_3.sizePolicy().hasHeightForWidth())
221     self.widget_3.setSizePolicy(sizePolicy)
222     self.widget_3.setAutoFillBackground(False)
223     self.widget_3.setObjectName("widget_3")
224     self.glasses = QtWidgets.QFrame(self.widget_3)
```

The screenshot shows the PyCharm IDE interface with the following details:

- Title Bar:** interface.py, Version control
- File Explorer:** Shows the project structure under Scratches and Consoles, including C:\Users\deevi\PyCharmCE2023.1\Scratches with files scratch.py, scratch_1.py, scratch_2.py, and scratch_3.py.
- Code Editor:** The interface.py file is open, displaying Python code for a Qt application. The code includes imports from QtWidgets, QtCore, and QtGui, and defines a class UI_MainWindow with a setupUI method. The code uses QFrame, QLabel, QPushButton, and QFont classes to create UI elements like glasses frames and buttons.
- Status Bar:** 251:27 CRLF UTF-8 8 spaces* Python 3.8

```
222     self.widget_3.setAutoFillBackground(False)
223     self.widget_3.setObjectName("widget_3")
224     self.glasses = QtWidgets.QFrame(self.widget_3)
225     self.glasses.setGeometry(QtCore.QRect(0, 0, 791, 291))
226     sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Preferred)
227     sizePolicy.setHorizontalStretch(0)
228     sizePolicy.setVerticalStretch(0)
229     sizePolicy.setHeightForWidth(self.glasses.sizePolicy().hasHeightForWidth())
230     self.glasses.setSizePolicy(sizePolicy)
231     self.glasses.setAutoFillBackground(False)
232     self.glasses.setFrameShape(QtWidgets.QFrame.StyledPanel)
233     self.glasses.setFrameShadow(QtWidgets.QFrame.Raised)
234     self.glasses.setObjectName("glasses")
235     self.widget_5 = QtWidgets.QWidget(self.glasses)
236     self.widget_5.setGeometry(QtCore.QRect(30, 50, 141, 171))
237     self.widget_5.setObjectName("widget_5")
238     self.label_7 = QtWidgets.QLabel(self.widget_5)
239     self.label_7.setGeometry(QtCore.QRect(0, 10, 140, 141))
240     self.label_7.setText("")
241     self.label_7.setPixmap(QtGui.QPixmap(":/blueIcons/glass1.png"))
242     self.label_7.setScaledContents(True)
243     self.label_7.setObjectName("label_7")
244     self.widget_7 = QtWidgets.QWidget(self.glasses)
245     self.widget_7.setGeometry(QtCore.QRect(190, 50, 141, 171))
246     self.widget_7.setObjectName("widget_7")
247     self.label_3 = QtWidgets.QLabel(self.widget_7)
248     self.label_3.setGeometry(QtCore.QRect(0, 20, 140, 141))
249     self.label_3.setText("")
250     self.label_3.setPixmap(QtGui.QPixmap(":/blueIcons/glass.png"))
251     self.label_3.setScaledContents(True)
252     self.label_3.setObjectName("label_3")
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
```

The screenshot shows the PyCharm IDE interface with the following details:

- Title Bar:** interface.py, Version control
- File Explorer:** Shows the project structure under Scratches and Consoles, including C:\Users\deevi\PyCharmCE2023.1\Scratches with files scratch.py, scratch_1.py, scratch_2.py, and scratch_3.py.
- Code Editor:** The interface.py file is open, displaying Python code for a Qt application. The code includes imports from QtWidgets, QtCore, and QtGui, and defines a class UI_MainWindow with a setupUI method. The code uses QFrame, QLabel, QPushButton, and QFont classes to create UI elements like glasses frames and buttons.
- Status Bar:** 281:14 CRLF UTF-8 8 spaces* Python 3.8

```
251     self.label_8.setScaledContents(True)
252     self.label_8.setObjectName("label_8")
253     self.widget_4 = QtWidgets.QWidget(self.glasses)
254     self.widget_4.setGeometry(QtCore.QRect(370, 50, 141, 171))
255     self.widget_4.setObjectName("widget_4")
256     self.label_9 = QtWidgets.QLabel(self.widget_4)
257     self.label_9.setGeometry(QtCore.QRect(0, 20, 140, 141))
258     self.label_9.setText("")
259     self.label_9.setPixmap(QtGui.QPixmap(":/blueIcons/glass2.png"))
260     self.label_9.setScaledContents(True)
261     self.label_9.setObjectName("label_9")
262     self.label_2 = QtWidgets.QLabel(self.glasses)
263     self.label_2.setGeometry(QtCore.QRect(40, 10, 151, 20))
264     font = QtGui.QFont()
265     font.setPointSize(12)
266     font.setBold(True)
267     font.setWeight(75)
268     self.label_2.setFont(font)
269     self.label_2.setObjectName("label_2")
270     self.pushButton = QtWidgets.QPushButton(self.glasses)
271     self.pushButton.setGeometry(QtCore.QRect(50, 240, 93, 28))
272     self.pushButton.setObjectName("pushButton")
273     self.pushButton_2 = QtWidgets.QPushButton(self.glasses)
274     self.pushButton_2.setGeometry(QtCore.QRect(220, 240, 93, 28))
275     self.pushButton_2.setObjectName("pushButton_2")
276     self.pushButton_3 = QtWidgets.QPushButton(self.glasses)
277     self.pushButton_3.setGeometry(QtCore.QRect(400, 240, 93, 28))
278     self.pushButton_3.setObjectName("pushButton_3")
279     self.widget_8 = QtWidgets.QWidget(self.glasses)
280     self.widget_8.setGeometry(QtCore.QRect(540, 50, 141, 171))
281     self.widget_8.setStyleSheet("#widget_8\n"
```

Screenshot of PyCharm interface showing the code editor and file browser. The code editor displays Python code for a Qt application, specifically the `setupUi()` method of `UI_MainWindow`. The file browser on the left shows the project structure under `C:\Users\deevi`, including `Scratches and Consoles`, `AppData`, `Roaming`, `JetBrains`, and `PyCharmCE2023.1` with files `scratch.py`, `scratch_1.py`, `scratch_2.py`, and `scratch_3.py`. The status bar at the bottom indicates the file is `interface.py`, encoding is `UTF-8`, and Python version is `3.8`.

```
281     self.widget_8.setStyleSheet("#widget_8{\n"
282         "background-color: rgb(255, 255, 255);\n"
283         "border-radius:25px;}")
284
285     self.widget_8.setObjectName("widget_8")
286     self.label_10 = QtWidgets.QLabel(self.widget_8)
287     self.label_10.setGeometry(QtCore.QRect(0, 20, 140, 141))
288     self.label_10.setText("")
289     self.label_10.setPixmap(QtGui.QPixmap(":/blueIcons/glass3.png"))
290     self.label_10.setScaledContents(True)
291     self.label_10.setObjectName("label_10")
292     self.pushButton_4 = QtWidgets.QPushButton(self.glasses)
293     self.pushButton_4.setGeometry(QtCore.QRect(560, 240, 93, 28))
294     self.pushButton_4.setAutoFillBackground(False)
295     self.pushButton_4.setStyleSheet("#pushButton_4{\n"
296         "background-color: rgb(0, 150, 195);\n"
297         "color:white;\n"
298         "\n"
299         "border-radius:10px;\n"
300         "\n"
301     }")
302     self.pushButton_4.setObjectName("pushButton_4")
303     self.accessories = QtWidgets.QFrame(self.widget_3)
304     self.accessories.setGeometry(QtCore.QRect(0, 280, 791, 271))
305     self.accessories.setSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
306     self.accessories.setHorizontalStretch(0)
307     self.accessories.setVerticalStretch(0)
308     self.accessories.setHeightForWidth(self.accessories.sizePolicy().hasHeightForWidth())
309     self.accessories.setSizePolicy(sizePolicy)
310     self.accessories.setMaximumSize(QtCore.QSize(791, 271))
311     self.accessories.setAutoFillBackground(False)
312     self.accessories setFrameShape(QtWidgets.QFrame.StyledPanel)
313     self.accessories setFrameShadow(QtWidgets.QFrame.Raised)
```

Screenshot of PyCharm interface showing the code editor and file browser. The code editor displays Python code for a Qt application, specifically the `setupUi()` method of `UI_MainWindow`. The file browser on the left shows the project structure under `C:\Users\deevi`, including `Scratches and Consoles`, `AppData`, `Roaming`, `JetBrains`, and `PyCharmCE2023.1` with files `scratch.py`, `scratch_1.py`, `scratch_2.py`, and `scratch_3.py`. The status bar at the bottom indicates the file is `interface.py`, encoding is `UTF-8`, and Python version is `3.8`.

```
308     self.accessories.setMaximumSize(QtCore.QSize(791, 271))
309     self.accessories.setAutoFillBackground(False)
310     self.accessories setFrameShape(QtWidgets.QFrame.StyledPanel)
311     self.accessories setFrameShadow(QtWidgets.QFrame.Raised)
312     self.accessories.setObjectName("accessories")
313     self.label_4 = QtWidgets.QLabel(self.accessories)
314     self.label_4.setGeometry(QtCore.QRect(20, 10, 151, 16))
315     font = QtGui.QFont()
316     font.setPointSize(12)
317     font.setBold(True)
318     font.setWeight(75)
319     self.label_4.setFont(font)
320     self.label_4.setObjectName("label_4")
321     self.widget_9 = QtWidgets.QWidget(self.accessories)
322     self.widget_9.setGeometry(QtCore.QRect(30, 40, 140, 171))
323     self.widget_9.setMinimumSize(QtCore.QSize(140, 171))
324     self.widget_9.setAutoFillBackground(False)
325     self.widget_9.setStyleSheet("#widget_9{\n"
326         "background-color: rgb(255, 255, 255);\n"
327         "border-radius:25px;}")
328     self.widget_9.setObjectName("widget_9")
329     self.label_13 = QtWidgets.QLabel(self.widget_9)
330     self.label_13.setGeometry(QtCore.QRect(0, 20, 140, 141))
331     self.label_13.setText("")
332     self.label_13.setPixmap(QtGui.QPixmap(":/blueIcons/necklac1.png"))
333     self.label_13.setScaledContents(True)
334     self.label_13.setObjectName("label_13")
335     self.widget_10 = QtWidgets.QWidget(self.accessories)
336     self.widget_10.setGeometry(QtCore.QRect(210, 40, 141, 171))
337     self.widget_10.setStyleSheet("#widget_10{\n"
338         "background-color: rgb(255, 255, 255);\n"
```

A screenshot of the PyCharm IDE interface. The top bar shows "interface.py" and "Version control". The left sidebar displays a file tree under "Scratches and Consoles" with files like login.py, interface.py (the current file), SPLASH.py, and scratch_3.py. The main code editor area shows Python code for setting up UI elements. The status bar at the bottom right indicates the file is 363 lines long, uses CRLF line endings, is in UTF-8 encoding, has 8 spaces per indentation, and is written in Python 3.8.

```
337     self.widget_10.setStyleSheet("#widget_10{\n338     \"background-color: rgb(255, 255, 255);\\n\"\n339     \"border-radius:25px;\\n\"\n340\n341     self.widget_10.setObjectName("widget_10")\n342     self.label_14 = QtWidgets.QLabel(self.widget_10)\n343     self.label_14.setGeometry(QtCore.QRect(0, 20, 140, 141))\n344     self.label_14.setText("")\n345     self.label_14.setPixmap(QtGui.QPixmap(":/blueIcons/necklace_2.png"))\n346     self.label_14.setScaledContents(True)\n347     self.label_14.setObjectName("label_14")\n348     self.widget_11 = QtWidgets.QWidget(self.accessories)\n349     self.widget_11.setGeometry(QtCore.QRect(380, 40, 141, 171))\n350     self.widget_11.setStyleSheet("#widget_11{\n351     \"background-color: rgb(255, 255, 255);\\n\"\n352     \"border-radius:25px;\\n\"\n353     self.widget_11.setObjectName("widget_11")\n354     self.label_15 = QtWidgets.QLabel(self.widget_11)\n355     self.label_15.setGeometry(QtCore.QRect(0, 20, 140, 141))\n356     self.label_15.setText("")\n357     self.label_15.setPixmap(QtGui.QPixmap(":/blueIcons/necklace_3.png"))\n358     self.label_15.setScaledContents(True)\n359     self.label_15.setObjectName("label_15")\n360     self.widget_12 = QtWidgets.QWidget(self.accessories)\n361     self.widget_12.setGeometry(QtCore.QRect(550, 40, 141, 171))\n362     self.widget_12.setStyleSheet("#widget_12{\n363     \"background-color: rgb(255, 255, 255);\\n\"\n364     \"border-radius:25px;\\n\"\n365     self.widget_12.setObjectName("widget_12")\n366     self.label_16 = QtWidgets.QLabel(self.widget_12)\n367     self.label_16.setGeometry(QtCore.QRect(0, 20, 140, 141))\n368     self.label_16.setText("")
```

A screenshot of the PyCharm IDE interface, identical to the first one but showing more of the code. The code editor continues from the previous snippet, defining push buttons 5 through 7 with their respective styles, geometries, and object names. The status bar at the bottom right indicates the file is 391 lines long, uses CRLF line endings, is in UTF-8 encoding, has 8 spaces per indentation, and is written in Python 3.8.

```
362     \"background-color: rgb(255, 255, 255);\\n\"\n363     \"border-radius:25px;\\n\"\n364\n365     self.widget_12.setObjectName("widget_12")\n366     self.label_16 = QtWidgets.QLabel(self.widget_12)\n367     self.label_16.setGeometry(QtCore.QRect(0, 20, 140, 141))\n368     self.label_16.setText("")\n369     self.label_16.setPixmap(QtGui.QPixmap(":/blueIcons/necklace_4.png"))\n370     self.label_16.setScaledContents(True)\n371     self.label_16.setObjectName("label_16")\n372     self.pushButton_5 = QtWidgets.QPushButton(self.accessories)\n373     self.pushButton_5.setGeometry(QtCore.QRect(50, 230, 93, 28))\n374     self.pushButton_5.setStyleSheet("#pushButton_5{\n375     \"background-color: rgb(0, 150, 195);\\n\"\n376     \"color:white;\\n\"\n377     \"\\n\"\n378     \"border-radius:10px;\\n\"\n379     \"\\n\"\n380     self.pushButton_5.setObjectName("pushButton_5")\n381     self.pushButton_6 = QtWidgets.QPushButton(self.accessories)\n382     self.pushButton_6.setGeometry(QtCore.QRect(230, 230, 93, 28))\n383     self.pushButton_6.setStyleSheet("#pushButton_6{\n384     \"background-color: rgb(0, 150, 195);\\n\"\n385     \"color:white;\\n\"\n386     \"\\n\"\n387     \"border-radius:10px;\\n\"\n388     \"\\n\"\n389     self.pushButton_6.setObjectName("pushButton_6")\n390     self.pushButton_7 = QtWidgets.QPushButton(self.accessories)\n391     self.pushButton_7.setGeometry(QtCore.QRect(400, 230, 93, 28))\n392     self.pushButton_7.setStyleSheet("#pushButton_7{\n393     \"background-color: rgb(0, 150, 195);\\n\"\n394\n395     self.pushButton_7.setObjectName("pushButton_7")
```

Screenshot of PyCharm interface showing code for `interface.py`. The code is part of a Qt application, specifically the `setupUI()` method for `Ui_MainWindow`. The code handles the styling and configuration of several UI elements, including push buttons and labels.

```
398     self.pushButton_7.setGeometry(QtCore.QRect(400, 230, 93, 28))
399     self.pushButton_7.setStyleSheet("#pushButton_7{\n"
400         "background-color: rgb(0, 150, 195);\n"
401         "color:white;\n"
402         "\n"
403         "border-radius:10px;\n"
404     }")
405
406     self.pushButton_7.setObjectName("pushButton_7")
407     self.pushButton_8 = QtWidgets.QPushButton(self.accessories)
408     self.pushButton_8.setGeometry(QtCore.QRect(580, 230, 93, 28))
409     self.pushButton_8.setStyleSheet("#pushButton_8{\n"
410         "background-color: rgb(0, 150, 195);\n"
411         "color:white;\n"
412         "\n"
413         "border-radius:10px;\n"
414     }")
415
416     self.pushButton_8.setObjectName("pushButton_8")
417     self.widget_6 = QtWidgets.QWidget(self.widget_5)
418     self.widget_6.setGeometry(QtCore.QRect(800, 0, 251, 571))
419     self.widget_6.setMinimumSize(QtCore.QSize(241, 571))
420     self.widget_6.setAutoFillBackground(False)
421     self.widget_6.setObjectName("widget_6")
422     self.widget_13 = QtWidgets.QWidget(self.widget_6)
423     self.widget_13.setGeometry(QtCore.QRect(0, 10, 251, 261))
424     self.widget_13.setAutoFillBackground(False)
425     self.widget_13.setObjectName("widget_13")
426     self.label_17 = QtWidgets.QLabel(self.widget_13)
427     self.label_17.setGeometry(QtCore.QRect(0, 0, 251, 261))
428     self.label_17.setText("")
429     self.label_17.setPixmap(QtGui.QPixmap(":/blueIcons/model3.png"))
430     self.label_17.setScaledContents(True)
431
432     self.widget_14 = QtWidgets.QWidget(self.widget_6)
433     self.widget_14.setGeometry(QtCore.QRect(0, 290, 251, 261))
434     self.widget_14.setAutoFillBackground(False)
435     self.widget_14.setObjectName("widget_14")
436     self.label_18 = QtWidgets.QLabel(self.widget_14)
437     self.label_18.setGeometry(QtCore.QRect(0, 0, 251, 261))
438     self.label_18.setText("")
439     self.label_18.setPixmap(QtGui.QPixmap(":/blueIcons/model.jpg"))
440     self.label_18.setScaledContents(True)
441     self.label_18.setObjectName("label_18")
442     self.verticalLayout_4.addWidget(self.widget_5)
443
444     MainWindow.setCentralWidget(self.centralwidget)
445
446     self.retranslateUi(MainWindow)
447     QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

Screenshot of PyCharm interface showing code for `interface.py`. The code is part of a Qt application, specifically the `retranslateUI()` method for `MainWindow`. The code translates various UI strings using `QtCore.QCoreApplication.translate`.

```
418     self.label_17.setText("")
419     self.label_17.setPixmap(QtGui.QPixmap(":/blueIcons/model3.png"))
420     self.label_17.setScaledContents(True)
421     self.label_17.setObjectName("label_17")
422     self.widget_14 = QtWidgets.QWidget(self.widget_6)
423     self.widget_14.setGeometry(QtCore.QRect(0, 290, 251, 261))
424     self.widget_14.setAutoFillBackground(False)
425     self.widget_14.setObjectName("widget_14")
426     self.label_18 = QtWidgets.QLabel(self.widget_14)
427     self.label_18.setGeometry(QtCore.QRect(0, 0, 251, 261))
428     self.label_18.setText("")
429     self.label_18.setPixmap(QtGui.QPixmap(":/blueIcons/model.jpg"))
430     self.label_18.setScaledContents(True)
431     self.label_18.setObjectName("label_18")
432     self.verticalLayout_4.addWidget(self.widget_5)
433
434     MainWindow.setCentralWidget(self.centralwidget)
435
436     self.retranslateUi(MainWindow)
437     QtCore.QMetaObject.connectSlotsByName(MainWindow)
438
439     def retranslateUi(self, MainWindow):
440         _translate = QtCore.QCoreApplication.translate
441         MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
442         self.appheader.setText(_translate("MainWindow", "Virtual Try on"))
443         self.label.setText(_translate("MainWindow", "HOME"))
444         self.label_5.setText(_translate("MainWindow", "TRY ON"))
445         self.label_11.setText(_translate("MainWindow", "CONTACT US"))
446         self.label_2.setText(_translate("MainWindow", "GLASSES"))
447         self.pushButton.setText(_translate("MainWindow", "Try on"))
448         self.pushButton_2.setText(_translate("MainWindow", "Try on"))
449         self.pushButton_3.setText(_translate("MainWindow", "Try on"))
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays a file tree under 'Scratches and Consoles'. The main editor window shows the 'interface.py' file with code related to a splash screen and window styling. The status bar at the bottom indicates the file is 'interface.py' with line 476 selected, and the bottom right shows Python 3.8.

```
448     self.pushButton_3.setText(_translate("MainWindow", "Try on"))
449     self.pushButton_4.setText(_translate("MainWindow", "Try on"))
450     self.label_4.setText(_translate("MainWindow", "ACCESSORIES"))
451     self.pushButton_5.setText(_translate("MainWindow", "Try on"))
452     self.pushButton_6.setText(_translate("MainWindow", "Try on"))
453     self.pushButton_7.setText(_translate("MainWindow", "Try on"))
454     self.pushButton_8.setText(_translate("MainWindow", "Try on"))
455     import resources
456
457
458 if __name__ == "__main__":
459     import sys
460     app = QtWidgets.QApplication(sys.argv)
461     MainWindow = QtWidgets.QMainWindow()
462     splash = SplashScreen()
463     splash.show()
464     splash.setStyleSheet(''' #LabelTitle {
465         font-size: 60px;
466         color: #93deed;
467     }
468
469     #LabelDesc {
470         font-size: 30px;
471         color: #c2ced1;
472     }
473
474     #LabelLoading {
475         font-size: 30px;
476         color: #e8e8e8;
477     }
478
479
480 if __name__ == "__main__":
481     import sys
482     app = QtWidgets.QApplication(sys.argv)
483     MainWindow = QtWidgets.QMainWindow()
```

The screenshot shows the PyCharm IDE interface. The top bar displays 'interface.py' and 'Version control'. The left sidebar shows a tree view of 'Scratches and Consoles' under 'C:\Users\deevi'. The main area is a code editor with the following content:

```
476     color: #e8e8e8;
477 }
478
479 QFrame {
480     background-color: #2F4454;
481     color: rgb(220, 220, 220);
482 }
483
484 QProgressBar {
485     background-color: #0A7B93;
486     color: rgb(200, 200, 200);
487     border-style: none;
488     border-radius: 10px;
489     text-align: center;
490     font-size: 30px;
491 }
492
493 QProgressBar::chunk {
494     border-radius: 10px;
495     background-color: qlineargradient(spread:pad x1:0, x2:1, y1:0.511364, y2:0.623, stop:0 #1C3334, stop:1 #376E6F);
496 }
497 ...
498 }
499 ui = Ui_MainWindow()
500 ui.setupUi(MainWindow)
501 timer = QtCore.QTimer()
502 timer.timeout.connect(lambda: MainWindow.show())
503 timer.start(10000)
504 sys.exit(app.exec_())
505
```

@Splashscreen

The screenshot shows the PyCharm IDE interface with the file `splash.py` open. The code implements a splash screen using PyQt5. It defines a class `SplashScreen` that initializes a window with a title, sets its size and flags, and starts a timer to update its content over time. The code uses `QLabel` for text, `QProgressBar` for progress, and `QFrame` for the main container.

```
import time
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QWidget, QProgressBar, QLabel, QFrame, QVBoxLayout, QHBoxLayout
from PyQt5.QtCore import Qt, QTimer
import sys

class SplashScreen(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Splash Screen Example')

        self.setFixedSize(1100, 500)
        self.setWindowFlag(Qt.FramelessWindowHint)
        self.setAttribute(Qt.WA_TranslucentBackground)

        self.counter = 0
        self.n = 300

        self.initUI()

        self.timer = QTimer()
        self.timer.timeout.connect(self.loading)
        self.timer.start(30)

    usage
    def initUI(self):
        layout = QVBoxLayout()
        self.setLayout(layout)

        self.frame = QFrame()
        layout.addWidget(self.frame)
```

The screenshot continues the code from the previous screenshot. It shows the implementation of the `initUI` method, which creates a `QFrame` and adds it to a `QVBoxLayout`. Inside the frame, it adds three labels: `self.labelTitle`, `self.labelDescription`, and `self.labelLoading`, and a `QProgressBar`. The labels are centered, and the progress bar is set to range from 0 to `n` with a value of 20.

```
self.labelTitle = QLabel(self.frame)
self.labelTitle.setObjectName('LabelTitle')

# center labels
self.labelTitle.resize(self.width() - 10, 150)
self.labelTitle.move(0, 40) # x, y
self.labelTitle.setText('Welcome')
self.labelTitle.setAlignment(Qt.AlignCenter)

self.labelDescription = QLabel(self.frame)
self.labelDescription.resize(self.width() - 10, 50)
self.labelDescription.move(0, self.labelTitle.height())
self.labelDescription.setObjectName('LabelDesc')
self.labelDescription.setText('<strong>Virtual try on</strong>')
self.labelDescription.setAlignment(Qt.AlignCenter)

self.progressBar = QProgressBar(self.frame)
self.progressBar.resize(self.width() - 200 - 10, 50)
self.progressBar.move(100, self.labelDescription.y() + 150)
self.progressBar.setAlignment(Qt.AlignCenter)
self.progressBar.setFormat('%p%')
self.progressBar.setTextVisible(True)
self.progressBar.setRange(0, self.n)
self.progressBar.setValue(20)

self.labelLoading = QLabel(self.frame)
self.labelLoading.resize(self.width() - 10, 50)
self.labelLoading.move(0, self.progressBar.y() + 70)
self.labelLoading.setObjectName('LabelLoading')
self.labelLoading.setAlignment(Qt.AlignCenter)
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays a tree view of the project structure under 'Scratches and Consoles'. The main area shows the content of the `splashh.py` file.

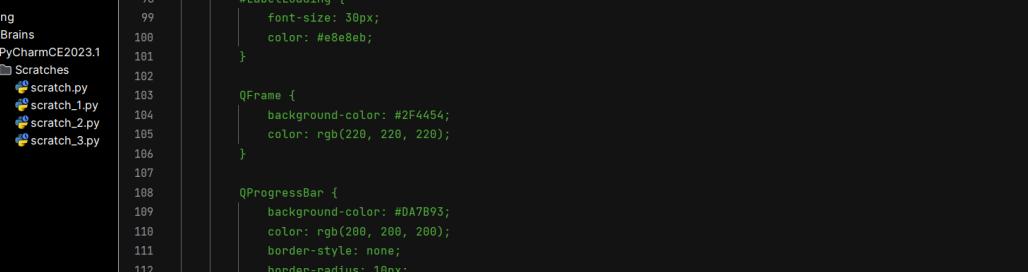
```
Scratches and Consoles ▾
  C:\Users\deevi
    AppData
    Roaming
      JetBrains
        PyCharmCE2023.1
          Scratches
            scratch.py
            scratch_1.py
            scratch_2.py
            scratch_3.py

login.py interface.py splashh.py SPLASH.py scratch_3.py

61 |     self.labelLoading.setAlignment(Qt.AlignCenter)
62 |     self.labelLoading.setText('loading...')
63 |
64     def loading(self):
65         self.progressBar.setValue(self.counter)
66
67         if self.counter == int(self.n * 0.3):
68             self.labelDescription.setText('<strong>Discover Your Perfect Look</strong>')
69         elif self.counter == int(self.n * 0.6):
70             self.labelDescription.setText('<strong>Transform Your Style</strong>')
71         elif self.counter >= self.n:
72             self.timer.stop()
73             self.close()
74
75             time.sleep(1)
76
77
78
79         self.counter += 1
80
81
82
83
84
85 > if __name__ == '__main__':
86     app = QApplication(sys.argv)
87     app.setStyleSheet('''
88         #LabelTitle {
89             font-size: 60px;
90             color: #93deed;
91         }
92     ''')

SplashScreen > __init__()

12:37 CRLF UTF-8 4 spaces Python 3.8
```



The screenshot shows the PyCharm IDE interface. The top navigation bar includes tabs for "interface.py" and "Version control". On the far right, there are icons for "util", "File", "Edit", "Search", and "Help". The left sidebar displays a file tree under "Scratches and Consoles" with a node for "C:\Users\deevi". The main editor area shows the content of "scratch_3.py". The code contains CSS-like styling for various Qt widgets like QLabel, QFrame, and QProgressBar, along with Python code for initializing a SplashScreen.

```
97     #LabelLoading {
98         font-size: 30px;
99         color: #e8e8eb;
100    }
101
102    QFrame {
103        background-color: #2F4454;
104        color: rgb(220, 220, 220);
105    }
106
107    QProgressBar {
108        background-color: #DA7B93;
109        color: rgb(200, 200, 200);
110        border-style: none;
111        border-radius: 10px;
112        text-align: center;
113        font-size: 30px;
114    }
115
116
117    QProgressBar::chunk {
118        border-radius: 10px;
119        background-color: qlineargradient(spread:pad x1:0, x2:1, y1:0.511364, y2:0.523, stop:0 #1C3334, stop:1 #376E6F);
120    }
121    ...
122
123    splash = SplashScreen()
124    splash.show()
125
126    sys.exit(app.exec_())
127
```

@Spectacle Try On

```
:> Users > gayathri > Desktop > eswari > specs try on.py
1 import cv2
2 import dlib
3
4
5 image = cv2.imread("lee.jpg",cv2.IMREAD_COLOR)
6
7 sunglasses = cv2.imread("glass2.png", cv2.IMREAD_UNCHANGED)
8
9 detector = dlib.get_frontal_face_detector()
10 predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
11 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
12
13 # Detect faces in the grayscale image
14 faces = detector(gray,1)
15
16 # Iterate over the detected faces
17 for face in faces:
18     # Estimate facial landmarks for the current face
19     landmarks = predictor(gray, face)
20     # Get the coordinates of relevant facial landmarks
21     nose_tip = (landmarks.part(29).x, landmarks.part(29).y)
22     left_ear = (landmarks.part(0).x, landmarks.part(0).y)
23
24     right_ear = (landmarks.part(16).x, landmarks.part(16).y)
25     left_eye = (landmarks.part(36).x, landmarks.part(36).y)
26     right_eye = (landmarks.part(45).x, landmarks.part(45).y)
27
```

```
# Calculate the dimensions of the sunglasses
sunglass_width = right_ear[0] - left_ear[0]
sunglass_height = int((sunglasses.shape[0] / sunglasses.shape[1]) * sunglass_width)

# Resize the sunglasses image
sunglasses_resized = cv2.resize(sunglasses, (sunglass_width, sunglass_height))

# Calculate the position for placing the sunglasses
x_offset = int(left_eye[0] - (sunglass_width * 0.18))
y_offset = int(nose_tip[1] - (sunglass_height * 0.7))

# Overlay the sunglasses on the image
for c in range(0, 3):
    image[y_offset:y_offset+sunglass_height, x_offset:x_offset+sunglass_width, c] = \
        sunglasses_resized[:, :, c] * (sunglasses_resized[:, :, 3] / 255.0) + \
        image[y_offset:y_offset+sunglass_height, x_offset:x_offset+sunglass_width, c] * (1.0 - sunglasses_resized[:, :, 3] / 255.0)

# Display the resulting image
cv2.imshow('Sunglass Effect', image)
cv2.waitKey(0)

# Release resources
cv2.destroyAllWindows()
```

@Necklace Try On

The screenshot shows the Visual Studio Code interface with the file `necklace.py` open. The code implements a function `necklace_overlay()` that loads two images, detects faces in the hero image, and overlays a necklace onto the detected faces based on their dimensions and position.

```
C:\> Users > gayathri > Desktop > eswari > necklace.py
1 import cv2
2 import dlib
3
4 def necklace_overlay():
5     # Load the hero image
6     image = cv2.imread("girl1.jpeg")
7
8     # Load the necklace image with transparency
9     necklaces = cv2.imread("necklace3.png", cv2.IMREAD_UNCHANGED)
10
11    # Initialize the face detector and facial landmark predictor
12    detector = dlib.get_frontal_face_detector()
13    predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
14
15    # Convert the hero image to grayscale for face detection
16    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
17
18    # Detect faces in the grayscale image
19    faces = detector(gray, 1)
20
21    # Iterate over the detected faces
22    for face in faces:
23        # Estimate facial landmarks for the current face
24        landmarks = predictor(gray, face)
25
26        # Get the coordinates of relevant facial landmarks
27        nose_tip = (landmarks.part(30).x, landmarks.part(30).y)
28        left_ear = (landmarks.part(0).x, landmarks.part(1).y)
29        right_ear = (landmarks.part(14).x, landmarks.part(15).y)
30        left_eye = (landmarks.part(37).x, landmarks.part(36).y)
31        right_eye = (landmarks.part(43).x, landmarks.part(45).y)
32
33        # Calculate the dimensions of the necklace
34        necklace_width = right_ear[0] - left_ear[0]
35        necklace_height = int((necklaces.shape[0] / necklaces.shape[1]) * necklace_width)
36
37        # Resize the necklace image
38        necklaces_resized = cv2.resize(necklaces, (necklace_width, necklace_height))
39
40        # Calculate the position for placing the necklace
41        x_offset = int((left_eye[0] - (necklace_width * 0.2)) * 0.98)
42        y_offset = int((nose_tip[1] - (necklace_height * 0.2)) * 1.5)
43
44        # Overlay the necklace on the image
45        for c in range(0, 3):
46            image[y_offset:y_offset+necklace_height, x_offset:x_offset+necklace_width, c] = \
47                necklaces_resized[:, :, c] * (necklaces_resized[:, :, 3] / 255.0) + \
48                image[y_offset:y_offset+necklace_height, x_offset:x_offset+necklace_width, c] * (1.0 - necklaces_resized[:, :, 3] / 255.0)
49
50        # Display the resulting image
51        cv2.imshow('necklace Effect', image)
52        cv2.waitKey(0)
53
54    # Release resources
55    cv2.destroyAllWindows()
56
57 necklace_overlay()
```

Ln 21, Col 23 Spaces: 4 UTF-8 CRLF Python ⚙️

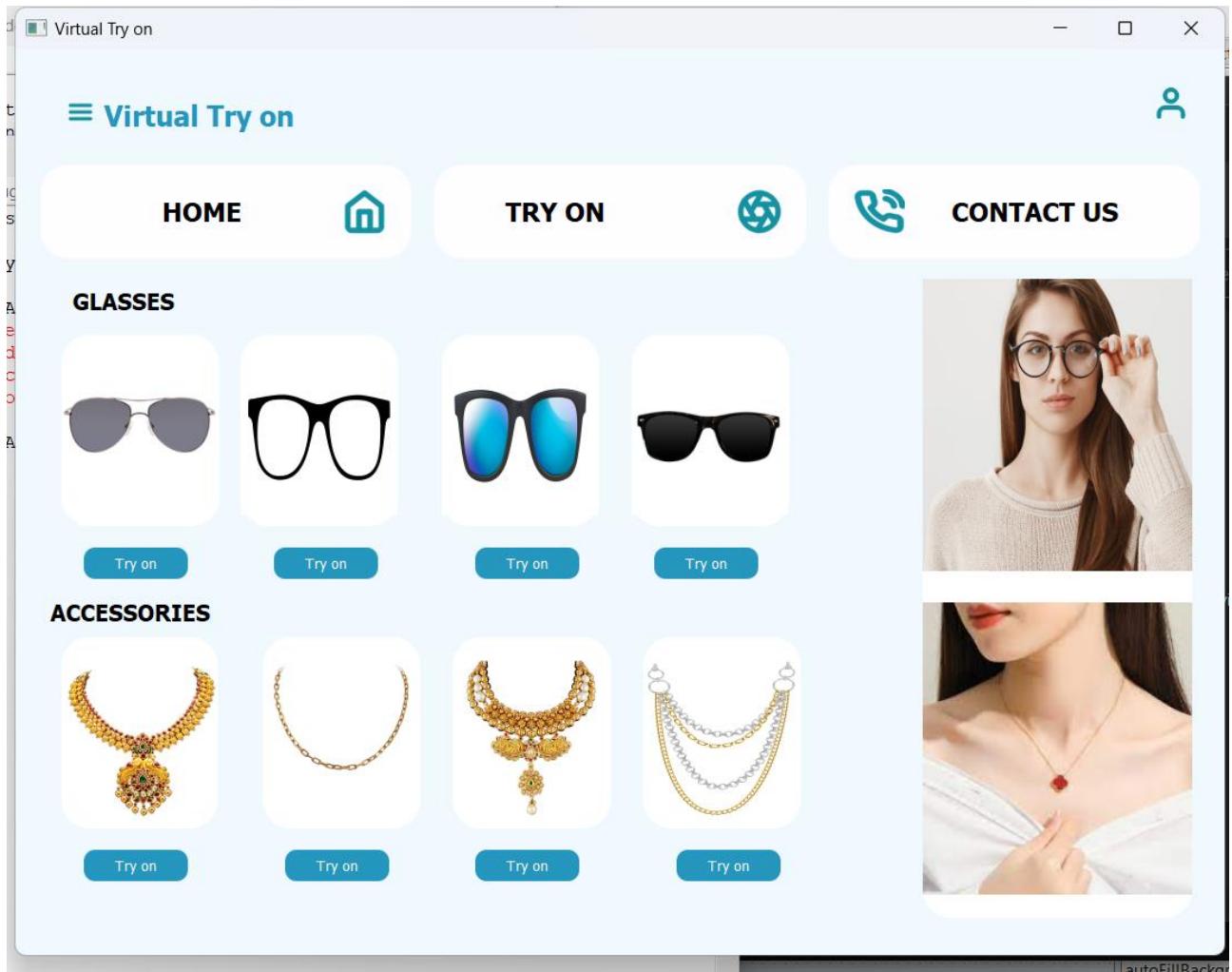
The screenshot shows the Visual Studio Code interface with the file `necklace.py` open. The code has been modified to include the logic for overlaying the necklace onto the detected faces. It uses a nested loop to iterate over each detected face and its corresponding landmarks to calculate the position and dimensions of the necklace and then overlay it onto the image.

```
C:\> Users > gayathri > Desktop > eswari > necklace.py
1 # Get the coordinates of relevant facial landmarks
2 nose_tip = (landmarks.part(30).x, landmarks.part(30).y)
3 left_ear = (landmarks.part(0).x, landmarks.part(1).y)
4 right_ear = (landmarks.part(14).x, landmarks.part(15).y)
5 left_eye = (landmarks.part(37).x, landmarks.part(36).y)
6 right_eye = (landmarks.part(43).x, landmarks.part(45).y)
7
8 # Calculate the dimensions of the necklace
9 necklace_width = right_ear[0] - left_ear[0]
10 necklace_height = int((necklaces.shape[0] / necklaces.shape[1]) * necklace_width)
11
12 # Resize the necklace image
13 necklaces_resized = cv2.resize(necklaces, (necklace_width, necklace_height))
14
15 # Calculate the position for placing the necklace
16 x_offset = int((left_eye[0] - (necklace_width * 0.2)) * 0.98)
17 y_offset = int((nose_tip[1] - (necklace_height * 0.2)) * 1.5)
18
19 # Overlay the necklace on the image
20 for c in range(0, 3):
21     image[y_offset:y_offset+necklace_height, x_offset:x_offset+necklace_width, c] = \
22         necklaces_resized[:, :, c] * (necklaces_resized[:, :, 3] / 255.0) + \
23         image[y_offset:y_offset+necklace_height, x_offset:x_offset+necklace_width, c] * (1.0 - necklaces_resized[:, :, 3] / 255.0)
24
25 # Display the resulting image
26 cv2.imshow('necklace Effect', image)
27 cv2.waitKey(0)
28
29 # Release resources
30 cv2.destroyAllWindows()
31
32 necklace_overlay()
```

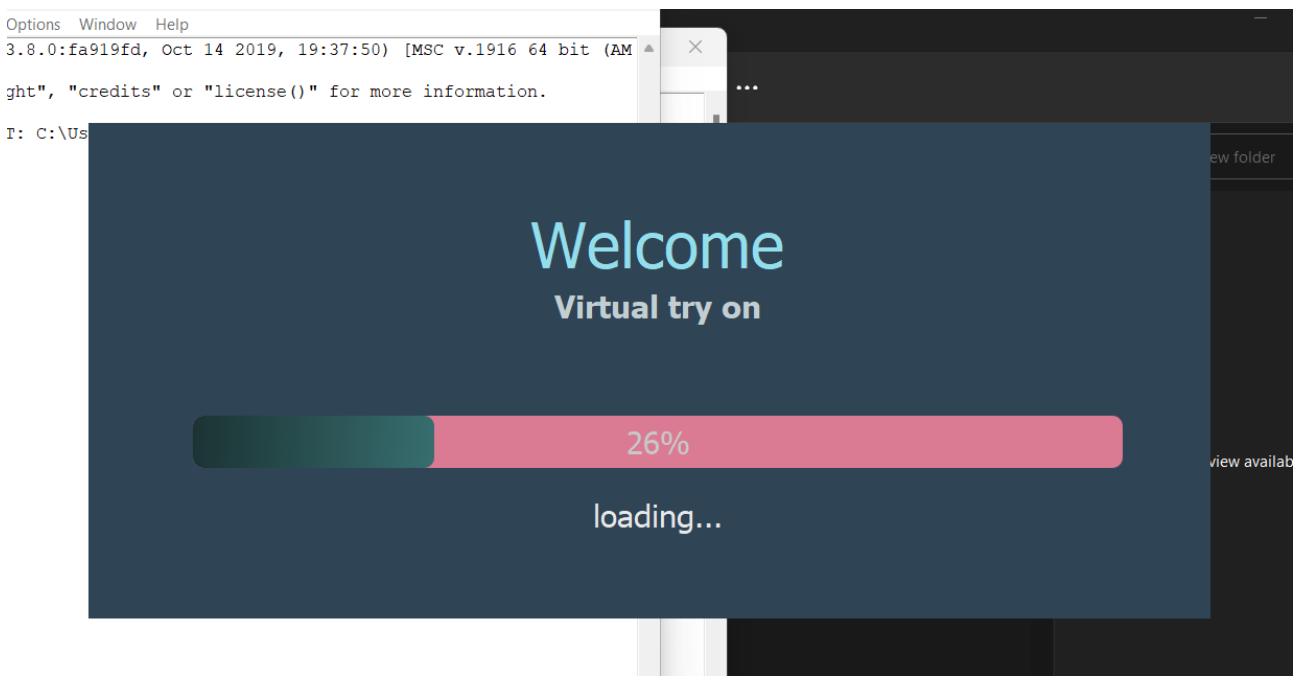
Ln 55, Col 19 Spaces: 4 UTF-8 CRLF Python ⚙️

5.OUTPUT SCREENSHOTS

@UserInterface



@SplashScreen



@Try On Glasses Output

Input images:



Output Image



@Necklace Try On

Input Images:



Output Image



6.CONCLUSION

In conclusion, virtual try-on using 2D image overlaying is a game changer for online shopping of glasses and necklaces. It helps solve the problem of buying these items without trying them on first, and provides an easy and convenient way to do so from the comfort of your own home.

By using this technology, you can save time and money, and have a better shopping experience overall. So next time you're in the market for new glasses or necklaces, give virtual try-on using 2D image overlaying a try. You won't be disappointed!

REFERENCES:

<https://chat.openai.com/>

<https://contribute.qt-project.org/>

<https://www.qt.io/>

[www.google.comm](http://www.google.com)

www.youtube.com (Code First with Hala, Jie Jenn)