Railway Sentry: Detecting Workers on Railway Tracks using YOLO V9

INTRODUCTION:

Railway Sentry is an advanced safety system that detects workers on railway tracks in real time using YOLO V9, the latest iteration of the "You Only Look Once" object detection algorithm. This model quickly identifies human figures on or near tracks, alerting operators to prevent accidents. By leveraging deep learning and computer vision, Railway Sentry enhances track safety, offering a proactive solution to reduce risks for railway maintenance crews.

Scenario 1:

Night Shift Surveillance:

Railway Sentry enhances worker safety on night shifts, when visibility is low. YOLO V9 detects human figures on tracks, even in dark or foggy conditions, by leveraging infrared and low-light cameras. Operators receive alerts if workers approach active tracks, mitigating hazards and providing additional security during low-visibility hours.

Scenario 2:

Routine Maintenance Detection:

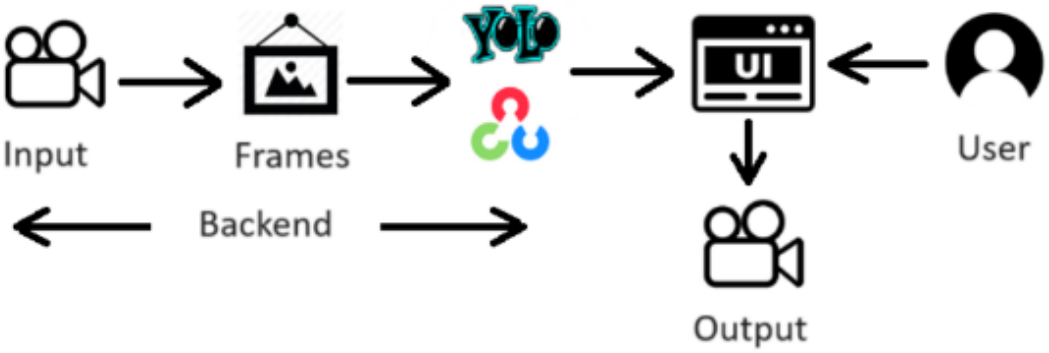
During scheduled track maintenance, Railway Sentry uses YOLO V9 to detect workers in designated safety zones. Alerts are issued if any worker strays into active track areas, allowing rapid intervention to ensure safety. This feature minimizes risks during standard operations by maintaining real-time awareness of crew positions and automatically notifying central control.

Scenario 3:

Emergency Response:

In cases where unexpected track repairs are needed, Railway Sentry can quickly identify emergency responders near live rails. The system highlights any worker movement on the track, enabling real-time monitoring and coordination. This immediate alerting function improves response times and safeguards teams during high-risk, unscheduled repairs.

Technical Architecture:



Pre-requisites:

To complete this project, you must require the following software, concepts, and packages.

1. IDE Installation:

Spyder/ PyCharm IDE is Ideal to complete this project

To install Spyder IDE, please refer to Spyder IDE Installation Steps

To install PyCharm IDE, please refer to the PyCharm IDE Installation Steps

2. Python Packages

If you are using the vs code, follow the below steps to download the required packages:

Open the vs code.

- Type “pip install ultralytics” and click enter.
- Type “pip install numpy ” and click enter
- Type “pip install flask” and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- Flask - https://www.youtube.com/watch?v=lj4I_CvBnt
- Yolov9 - https://youtu.be/ZF7EAodHn1U

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for computer vision.
- Gain knowledge in the pre-trained model yolov9.

Project Flow:

The user interacts with the UI (User Interface) to choose the image.

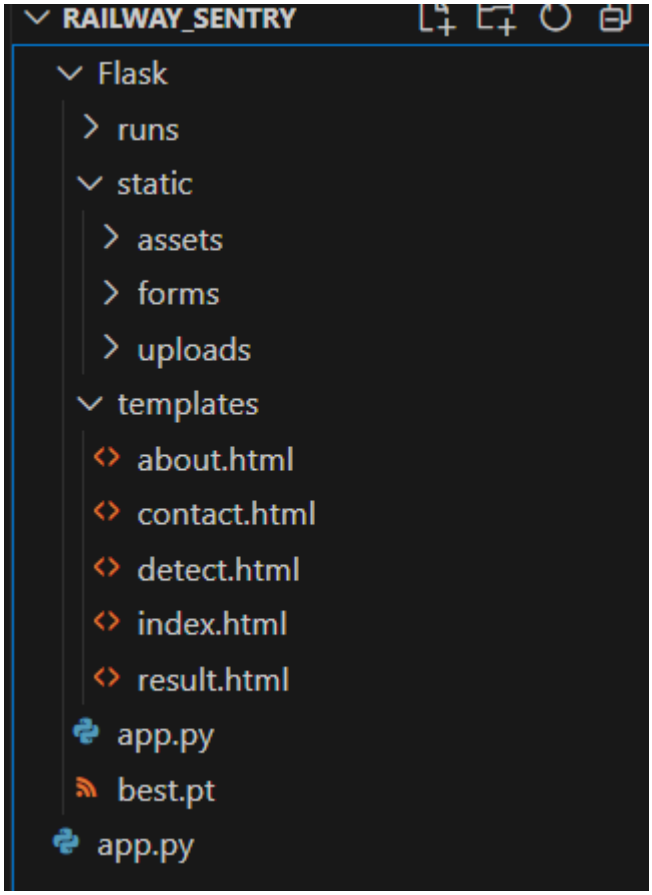
The chosen image is analyzed by the model which is integrated with the flask application.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
- Create Train and Test Folders.
- Create data.yaml file
- Training and testing the model
- Save the Model
- Application Building
- Create an HTML file
- Build Python Code

Project Structure:

Create a Project folder which contains files as shown below.



● The Dataset folder contains the training, testing, and val images for training our model.

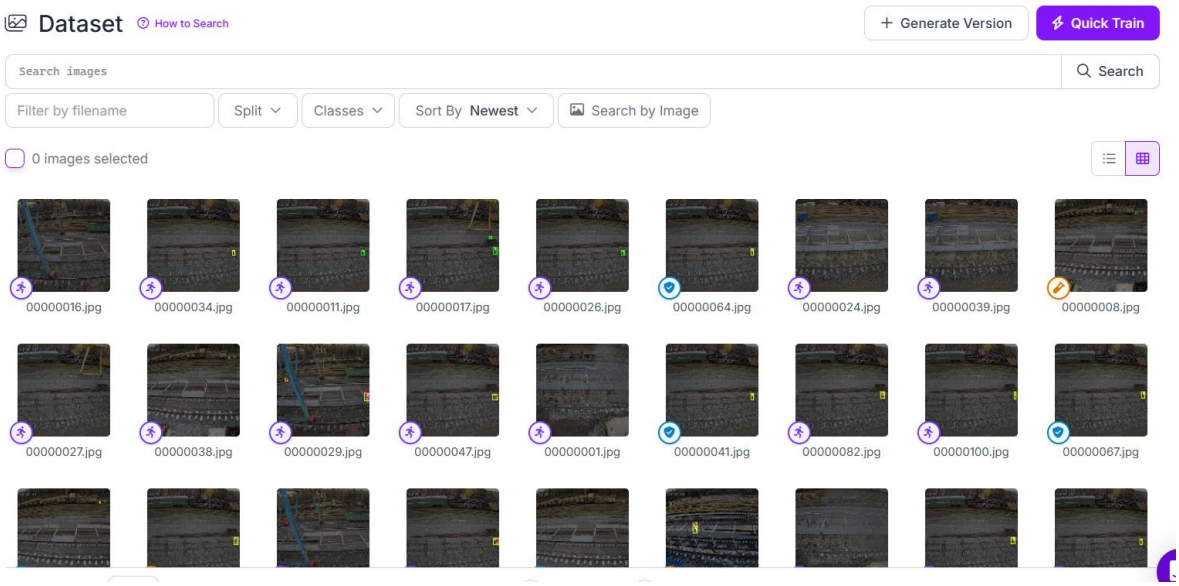
● We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server-side scripting

● We need the model which is saved and the saved model in this content there is a templates folder containing index.html and inner-page.html pages.

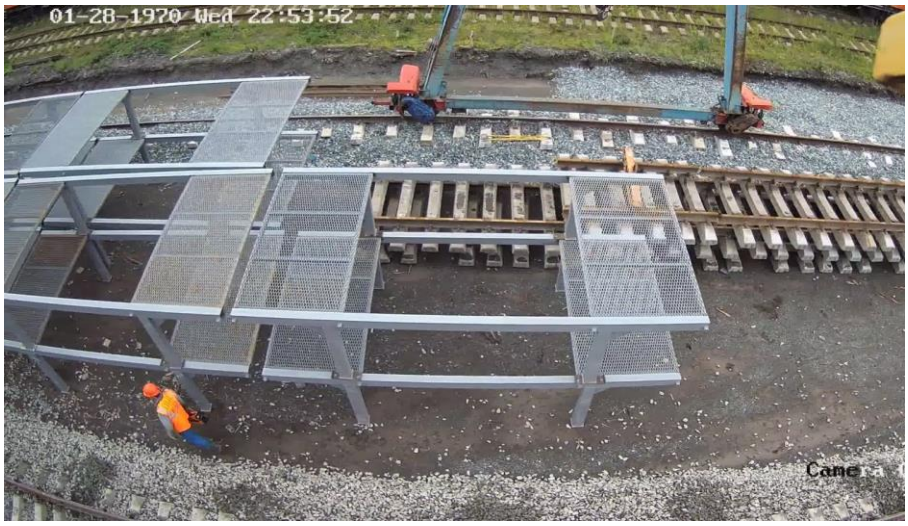
Milestone 1: Collection of Data

Dataset has 3 classes Which are Test, Train and Valid

Download the Dataset-



Sample Data:



Milestone 2: Image Pre-processing

In this milestone, we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Import the required libraries

We need to download yoloV9 from the ultralytics.

```
import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import seaborn as sns
import pathlib as path
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] import os
    os.chdir('/content/drive/MyDrive/Railway sentry.v1i.yolov9')
    os.listdir()
```

Activity 2: Download the pretrained weights

loading pre-trained model yolov9 weights from ultralytics

```
import locale
def getpreferredencoding(do_set=True):
    return "UTF-8"
locale.getpreferredencoding = getpreferredencoding
!wget https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov9s.pt
```

```
Show hidden output
```

```
[ ] from ultralytics import YOLO

    # Load a model
    model = YOLO("yolov9s.pt")  # load a pretrained model (recommended for training
```

Activity 3: Load the Dataset:

installing roboflow and connect our dataset from google drive

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
os.chdir('/content/drive/MyDrive/Railway sentry.v1i.yolov9')
os.listdir()
```

```
['yolov9s.pt.1',
 'yolov9s.pt',
 'README.roboflow.txt',
 'data.yaml',
 'README.dataset.txt',
 'valid',
 'train',
 'test',
 'runs',
 'yolo11n.pt']
```

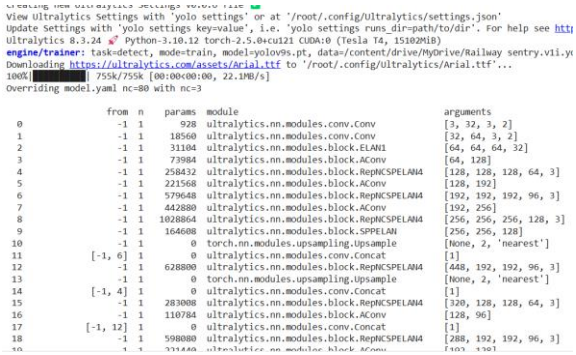
Milestone 3: training

Now it's time to train our Yolo model:

We have to create data.yaml:

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov9s.pt")  # load a pretrained model (recommended for training

# Train the model
results = model.train(data="/content/drive/MyDrive/Railway sentry.v1i.yolov9/data.yaml", epochs=100, imgsz=642)
```

Training yolo v9 model on a custom dataset.



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100      5.61G     0.6522     0.3492     0.8753          7        672: 100%|██████████| 43/43 [00:24<00:00,  1.73it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 6/6 [00:03<00:00,  1.50it/s]

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      5.55G     0.6669     0.3403     0.8758          3        672: 100%|██████████| 43/43 [00:23<00:00,  1.85it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 6/6 [00:02<00:00,  2.26it/s]

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      5.62G      0.662     0.3485     0.8707          2        672: 100%|██████████| 43/43 [00:25<00:00,  1.71it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 6/6 [00:02<00:00,  2.26it/s]

.00 epochs completed in 0.834 hours.
)ptimizer stripped from runs/detect/train4/weights/last.pt, 15.2MB
)ptimizer stripped from runs/detect/train4/weights/best.pt, 15.2MB

'alidating runs/detect/train4/weights/best.pt...
Ultralytics 8.3.24 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
'OLOv9s summary (fused): 486 layers, 7,168,249 parameters, 0 gradients, 26.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 6/6 [00:07<00:00,  1.33s/it]
                   all        192        879      0.906      0.907      0.951      0.701
                Helmet        147        350      0.878      0.865      0.931      0.593
                Person        190        529      0.934      0.949       0.97      0.809
;peed: 0.3ms preprocess, 9.2ms inference, 0.0ms loss, 7.9ms postprocess per image
!esults saved to runs/detect/train4
```

Validation:

Validating our model with a random image from the test folder. Also, we have saved our best.pt

```
results = model.val(data="/content/drive/MyDrive/Railway sentry.v1i.yolov9/data.yaml", epochs=80, imgsz=642)
```

```
WARNING ⚠️ imgsz=[642] must be multiple of max stride 32, updating to [672]
Ultralytics 8.3.24 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv9s summary (fused): 486 layers, 7,168,249 parameters, 0 gradients, 26.7 GFLOPs
val: Scanning /content/drive/MyDrive/Railway sentry.v1i.yolov9/valid/labels.cache... 192 images, 2 backgrounds
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/
                   all        192        879      0.906      0.905       0.95      0.708
                Helmet        147        350      0.878      0.863      0.932      0.604
                Person        190        529      0.935      0.947      0.969      0.811
Speed: 0.5ms preprocess, 14.1ms inference, 0.1ms loss, 4.1ms postprocess per image
Results saved to runs/detect/train42
```

Displaying detected image in training notebook:

```
# Path to your image
image_path = "/content/drive/MyDrive/Railway sentry.v1i.yolov9/test/ir

# Perform object detection
results = model(image_path)

# Access the first result (assuming only one image)
result = results[0]

# Get bounding boxes, class labels, and confidence scores
boxes = result.boxes.xyxy  # Bounding boxes (x1, y1, x2, y2)
labels = result.boxes.cls  # Class labels
confidences = result.boxes.conf  # Confidence scores

# Print the object representations
for box, label, confidence in zip(boxes, labels, confidences):
    print(f"Bounding Box: {box.tolist()}")
    print(f"Class Label: {label.item()}")
    print(f"Confidence: {confidence.item()}")

# Visualize the results with bounding boxes and labels
results[0].plot()  # This will display the image with the detected ob
```

Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1: Create HTML Pages

● We use HTML to create the front-end part of the web page.

● Here, we have created 3 HTML pages- home.html, intro.html, and upload.html

● home.html displays the home page.

● Intro.html displays an introduction about the project

● upload.html gives the emergency alert For more information regarding HTML

● We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

● Link:CSS, JS

Create app.py (Python Flask) file: -

Write the below code in Flask app.py python file script to run the Object Detection Project.

```
from flask import Flask, request, render_template, redirect, url_for, send_from_directory
import os
from ultralytics import YOLO
from PIL import Image
from werkzeug.utils import secure_filename

app = Flask(__name__)

# Define the path to the model
model_path = 'best.pt'

# Load the model
model = YOLO(model_path)


# model = YOLO('best.pt')

UPLOAD_FOLDER = 'static/uploads'
ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png', 'mp4', 'avi', 'mkv'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

To upload image in UI:

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/detect/', methods=['GET', 'POST'])
def detect():
    if request.method == 'POST':
        if 'file' not in request.files:
            return 'No file part'

        file = request.files['file']

        if file.filename == '':
            return "No selected file"
```

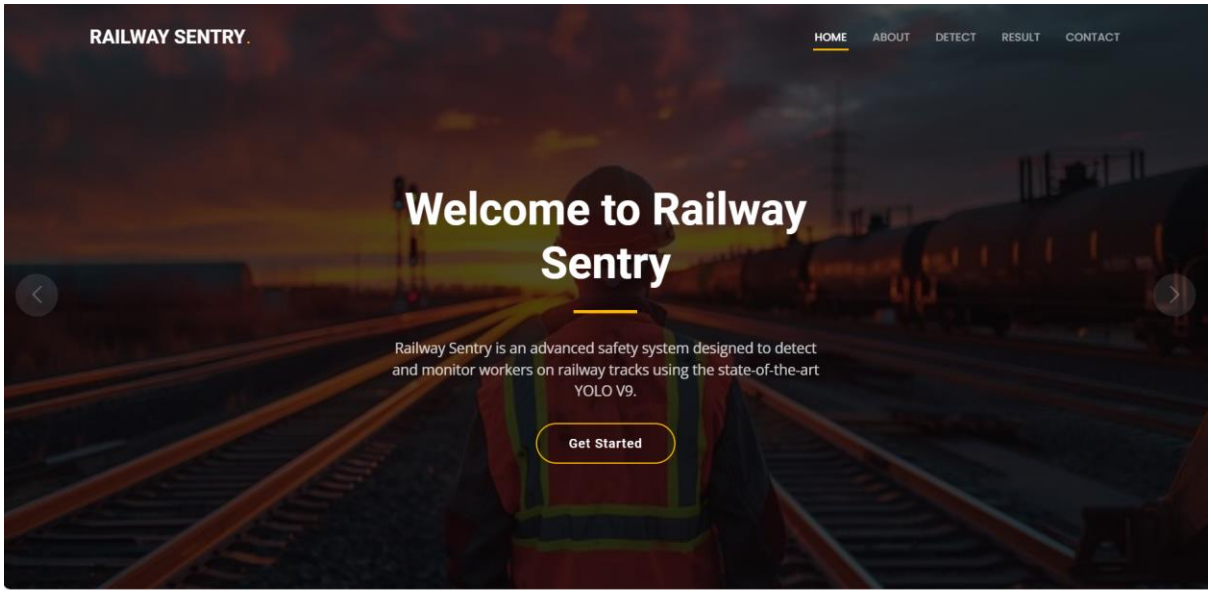
To display the image in UI:

```python
@app.route('/result/<original_filename>')
def result(original_filename):
    folder_path ='runs/detect'
    subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
    latest_subfolder = max(subfolders, key=lambda x: os.path.getctime(os.path.join(folder_path, x)))
    directory = folder_path+'/'+latest_subfolder
    print("printing directory:",directory)
    files = os.listdir(directory)
    latest_file = files[0]

    print(latest_file)

    filename = os.path.join(folder_path, latest_subfolder, latest_file)

    file_extension = filename.rsplit('.', 1)[1].lower()

    environ = request.environ
    if file_extension == 'jpg':
        return send_from_directory(directory,latest_file) #shows the result in seperate tab

    else:
        return "Invalid file format"

if __name__ == '__main__':
    app.run(debug=True)
```

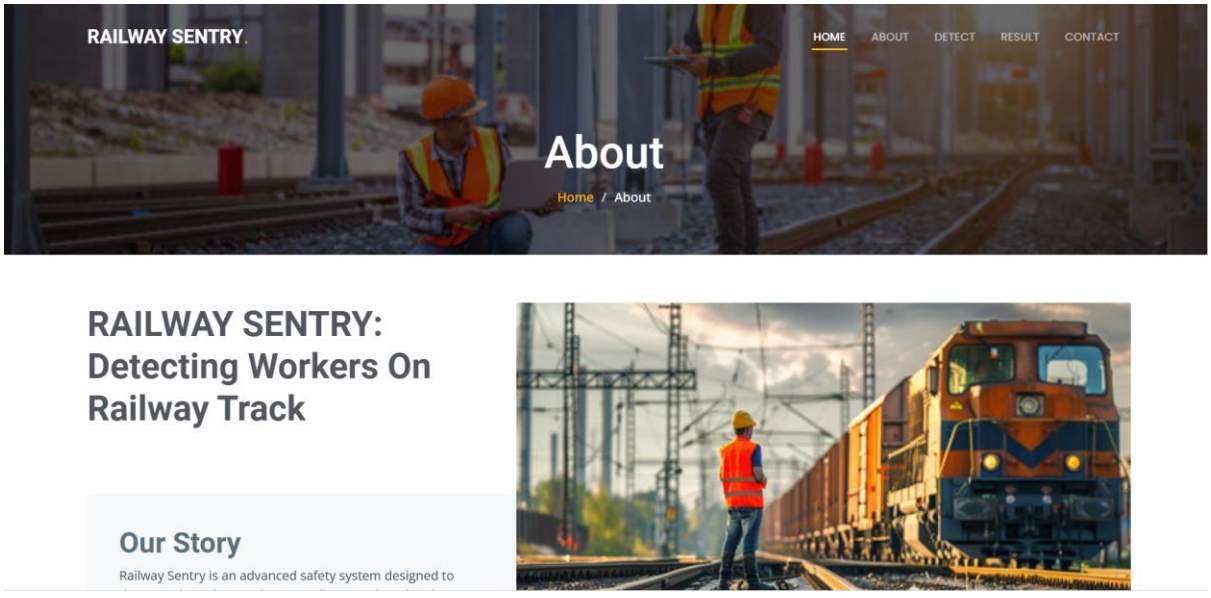
Getting local host in the terminal while running app.py:

```
PS C:\Users\prath\Downloads\Railway_sentry> cd Flask
PS C:\Users\prath\Downloads\Railway_sentry\Flask> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 145-983-214
```

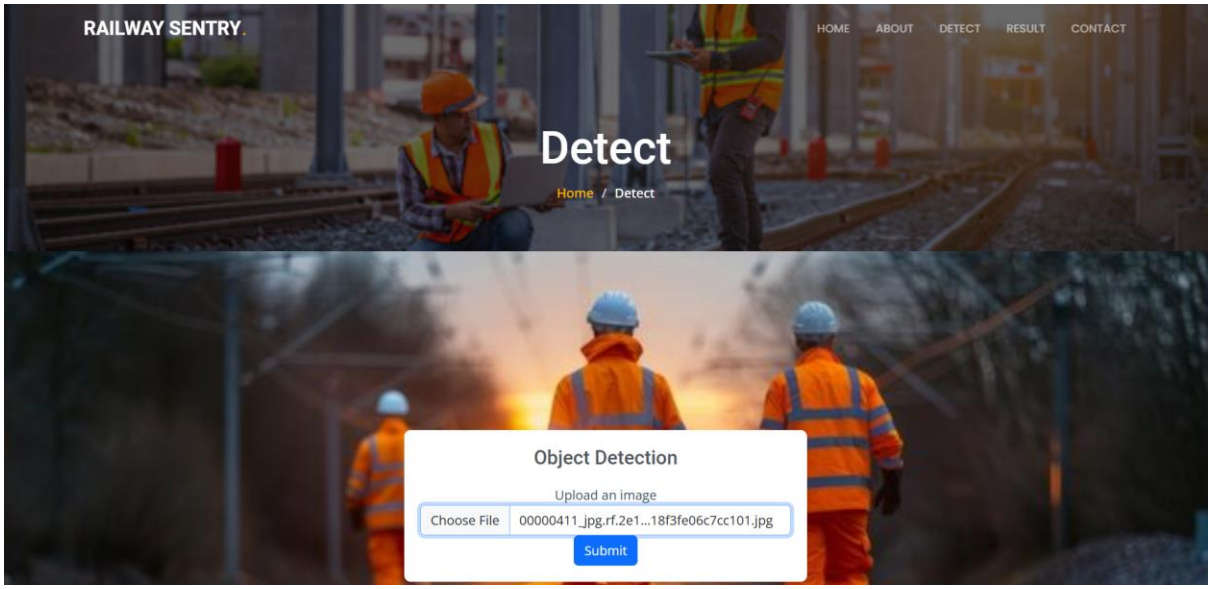
Index.html is displayed below:



About Section is displayed below:



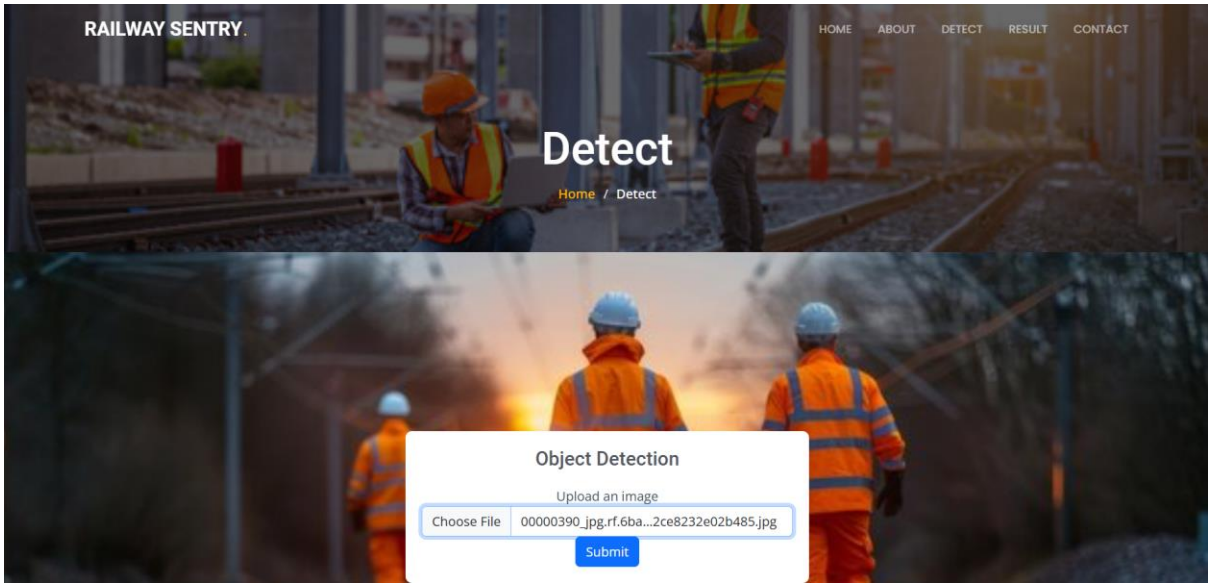
Upload and Output Page is displayed below Input:1



Final Output (after you click on Upload) is displayed as follows:

Output 1

Input:2



Output:2