Perfect! Here’s a **4-day structured and detailed development plan** for your ConsentScope web project on Ubuntu. Each day includes clear goals, tasks, commands, and expected outcomes. The plan assumes you're starting fresh, and I’ll guide you step-by-step.

# 🗓️ ****ConsentScope Development Plan (4 Days)****

## ✅ ****Day 1: Project Setup + Web App Skeleton + Initial Frontend****

### 🎯 Goal:

* Set up environment
* Create Flask web app skeleton
* Build the homepage with a URL input form

### 🔧 Tasks:

#### 1. Create the project folder and set up Python environment

mkdir consentscope

cd consentscope

python3 -m venv venv

source venv/bin/activate

pip install flask selenium selenium-wire beautifulsoup4 tldextract pandas matplotlib

#### 2. Set up the directory structure:

mkdir templates static engine logs

touch app.py

#### 3. Create the homepage (URL input form)

Inside templates/index.html:

<!DOCTYPE html>

<html>

<head>

<title>ConsentScope</title>

</head>

<body>

<h2>ConsentScope: Website Cookie Analysis</h2>

<form action="/analyze" method="POST">

<input type="text" name="url" placeholder="Enter website URL" required>

<button type="submit">Analyze</button>

</form>

</body>

</html>

#### 4. Create app.py base Flask app:

from flask import Flask, render\_template, request, redirect, url\_for

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/analyze', methods=['POST'])

def analyze():

url = request.form['url']

# For now, just pass it to a dummy result page

return f"<h3>Received: {url}</h3>"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

#### 5. Run the server:

python app.py

#### ✅ Output for Day 1:

* Working web page at http://127.0.0.1:5000/
* URL input form
* Flask app receives the URL and displays it

## ✅ ****Day 2: Selenium Cookie Consent Automation + Request Capture****

### 🎯 Goal:

* Install Chrome & ChromeDriver
* Launch site in Selenium
* Detect and click cookie banners
* Capture network requests

### 🔧 Tasks:

#### 1. Install Google Chrome:

wget https://dl.google.com/linux/direct/google-chrome-stable\_current\_amd64.deb

sudo apt install ./google-chrome-stable\_current\_amd64.deb

#### 2. Install ChromeDriver:

Check Chrome version:

google-chrome --version

Then download matching driver from:  
👉 <https://chromedriver.chromium.org/downloads>

Install it:

unzip chromedriver\_linux64.zip

chmod +x chromedriver

sudo mv chromedriver /usr/local/bin/

#### 3. Create engine/tracker\_analyzer.py:

from seleniumwire import webdriver

import time

import json

def analyze\_website(url, action="accept"):

options = webdriver.ChromeOptions()

options.add\_argument("--headless") # or remove this to see browser

driver = webdriver.Chrome(options=options)

driver.get(url)

time.sleep(5) # Wait for banner to appear

# Basic logic to click Accept/Reject

try:

if action == "accept":

buttons = driver.find\_elements("xpath", "//button[contains(text(), 'Accept')]")

else:

buttons = driver.find\_elements("xpath", "//button[contains(text(), 'Reject')]")

if buttons:

buttons[0].click()

time.sleep(5) # Wait for network requests after interaction

except Exception as e:

print("Cookie banner not handled:", e)

requests\_data = []

for request in driver.requests:

if request.response:

requests\_data.append({

"url": request.url,

"method": request.method,

"status": request.response.status\_code,

"headers": dict(request.headers)

})

driver.quit()

filename = f"logs/{url.replace('https://','').replace('/','\_')}\_{action}.json"

with open(filename, "w") as f:

json.dump(requests\_data, f, indent=2)

return filename

#### 4. Modify app.py to call it:

from engine.tracker\_analyzer import analyze\_website

@app.route('/analyze', methods=['POST'])

def analyze():

url = request.form['url']

accept\_log = analyze\_website(url, "accept")

reject\_log = analyze\_website(url, "reject")

return f"<h3>Analysis done. Accept log: {accept\_log}<br>Reject log: {reject\_log}</h3>"

#### ✅ Output for Day 2:

* Selenium loads site in headless mode
* Clicks Accept/Reject
* Captures all network requests
* Stores them in logs/ folder

## ✅ ****Day 3: Tracker Classification + Report Generation****

### 🎯 Goal:

* Classify third-party requests
* Generate tracker report (CSV or JSON)
* Display report in frontend

### 🔧 Tasks:

#### 1. Enhance tracker\_analyzer.py to:

* Use tldextract to compare domain
* Match against known tracker lists (load from CSV or JSON)

pip install tldextract

Update analyzer to return detected third-party requests:

import tldextract

def is\_third\_party(main\_domain, request\_url):

domain = tldextract.extract(request\_url).registered\_domain

return domain != main\_domain and domain != ''

def classify\_trackers(log\_path, main\_domain):

with open(log\_path) as f:

data = json.load(f)

trackers = []

for req in data:

url = req['url']

if is\_third\_party(main\_domain, url):

trackers.append(url)

return trackers

#### 2. Modify app.py to display report:

from flask import render\_template

from urllib.parse import urlparse

@app.route('/analyze', methods=['POST'])

def analyze():

url = request.form['url']

domain = urlparse(url).netloc

accept\_log = analyze\_website(url, "accept")

reject\_log = analyze\_website(url, "reject")

accept\_trackers = classify\_trackers(accept\_log, domain)

reject\_trackers = classify\_trackers(reject\_log, domain)

return render\_template("report.html", url=url, accept=accept\_trackers, reject=reject\_trackers)

#### 3. Create templates/report.html:

<h2>Analysis Report for {{ url }}</h2>

<h3>Trackers (Accept):</h3>

<ul>

{% for t in accept %}

<li>{{ t }}</li>

{% endfor %}

</ul>

<h3>Trackers (Reject):</h3>

<ul>

{% for t in reject %}

<li>{{ t }}</li>

{% endfor %}

</ul>

#### ✅ Output for Day 3:

* Page shows third-party trackers detected after Accept/Reject
* You’ve built the full data flow analysis loop!

## ✅ ****Day 4: Polish + Downloadable Report + Deployment Prep****

### 🎯 Goal:

* Add download buttons (CSV/JSON)
* Add basic styling
* Prepare for deployment (optional)

### 🔧 Tasks:

#### 1. Save CSV version of report:

In tracker\_analyzer.py:

import pandas as pd

def save\_csv(trackers, filename):

df = pd.DataFrame({"Tracker URL": trackers})

df.to\_csv(f"logs/{filename}.csv", index=False)

Call this from app.py after tracker classification:

save\_csv(accept\_trackers, f"{domain}\_accept")

save\_csv(reject\_trackers, f"{domain}\_reject")

#### 2. Add download links in report.html:

<a href="/static/{{ domain }}\_accept.csv" download>Download Accept CSV</a><br>

<a href="/static/{{ domain }}\_reject.csv" download>Download Reject CSV</a>

#### 3. Optional: Add some CSS in static/style.css and link it

#### 4. (Optional) Prepare for deployment:

* Create requirements.txt:

pip freeze > requirements.txt

* Set up Procfile for Render:

web: python app.py

## ✅ Final Output:

By Day 4, you’ll have:

* A working web app
* Accepts a URL
* Simulates cookie banner interaction
* Detects and classifies trackers
* Shows report + allows CSV downloads

Would you like me to:

* Start writing app.py and tracker\_analyzer.py files for you?
* Or generate a zip-ready template structure with boilerplate?

Let’s get building!