### **1. Introduction**

This document replaces and expands upon all prior handoff materials. It is intended to ensure the Replit agent builds exactly what is required—no deviations—covering:

1. The **Website Analyzer** (OTO2)
2. The **Site Builder Prompt** (used by OTO2 to generate a two‐page site with /index.html, /blog.html, and an empty /posts/ folder, for full OTO3 compatibility)

Every requirement, file, and behavior is enumerated below. Please follow this spec precisely.

### **2. Project Structure & Files**

5mm-website-analyzer/

│

├─ .replit # Optional; ensure it just runs `streamlit run main.py`

├─ requirements.txt # Dependencies

├─ authorized\_users.py # OPTIONAL: whitelist of allowed emails

├─ main.py # Streamlit application entry point

├─ analyzer\_utils.py # Helper functions: scraping, scoring, GPT calls

├─ replit\_prompt\_template.txt # The exact site‐builder prompt template (with placeholders)

├─ pdf\_generator.py # (Optional) PDF export logic

├─ README.md # Fork, setup, and run instructions for end users

└─ .gitignore # Exclude `.env`, `\_\_pycache\_\_/`, etc.

### **2.1 requirements.txt**

graphql

CopyEdit

streamlit

openai

requests

beautifulsoup4

python-dotenv

pdfkit # optional, only if PDF export is desired

### **2.2 .gitignore**

bash

CopyEdit

.env

\_\_pycache\_\_/

\*.pyc

### **2.3 authorized\_users.py (Optional Login Whitelist)**

python

CopyEdit

# List of authorized emails, all lowercase

AUTHORIZED\_EMAILS = [

"user1@example.com",

"user2@example.com",

# Add additional addresses as needed

]

### **2.4 replit\_prompt\_template.txt**

**Exact, verbatim site‐builder prompt with placeholders**—copy/paste this into the “Replit Prompt” area in the UI (Streamlit) so end users can copy it exactly. Do not modify formatting or wording; replace only the placeholders below at runtime.

Create a clean, mobile-responsive, two-page website for the following local business:

• Business Name: [Extract from https://{domain} or “Client Name”]

• Business Type: {business\_type} (e.g., “Plumbing Company”)

• Location: {location} (e.g., “Dallas, TX”)

Using the existing live site at https://{domain}, pull in all relevant content:

– Main headline (H1) or business name

– Service descriptions

– Contact details (phone, address, email)

– Logo or brand colors if available; if no logo exists, create a text-based logo using a modern font.

\*\*Folder/File structure:\*\*

/

├── index.html

├── blog.html

└── posts/ # ← empty folder; blog posts will go here later

---

### 1. Build `index.html` (Homepage)

Structure and content:

1. \*\*Hero Section\*\*

– Business name or slogan as a large H1.

– Single call-to-action button (“Call Now” or “Get a Free Quote”) with `tel:` link.

– Background or overlay image (use stock placeholder if none exists).

2. \*\*About Section\*\*

– Short paragraph (2–3 sentences) describing the business or owner’s background (pull from existing “About Us”).

– A bullet list “Why Choose Us” (e.g., “Licensed & Insured,” “24/7 Emergency Service,” “Over 15 Years of Experience”).

3. \*\*Services Section\*\*

– List core services (e.g., “Drain Cleaning,” “Water Heater Repair”).

– For each service, include a 1–2 sentence description, pulled from the existing site.

4. \*\*Testimonials / Trust Signals (Optional)\*\*

– If current site has customer reviews or star ratings, include 1–2. If none, omit.

5. \*\*Contact Section\*\*

– Address (plain text), clickable phone number `<a href="tel:…">`.

– Simple contact form placeholder (Name, Email/Phone, Message)—styled but non-functional.

6. \*\*Footer\*\*

– “© [Current Year] [Business Name]”

– Simple navigation links: Home, Blog, Services, Contact.

\*\*Design Guidelines for `index.html`:\*\*

• Use a modern sans-serif font (e.g., Montserrat or Open Sans).

• Keep a light background and dark text for legibility.

• Accent color: match existing brand color or use a complementary blue/teal if none exists.

• CTA button must stand out (high-contrast).

• All text size ≥ 16 px on mobile.

• Use semantic HTML5 tags only (`<section>`, `<header>`, `<footer>`, `<nav>`). Avoid deprecated tags (`<font>`, `<center>`, `<table>` for layout).

---

### 2. Build `blog.html` (Blog Index Page)

Structure and content:

1. \*\*Header\*\*

– `<h1>Blog</h1>`

– Subtitle: `<p>Latest news, tips, and updates from [Business Name]</p>`

2. \*\*Post Listing\*\*

– Use `<ol style="list-style: none; padding: 0;">`.

– Initially, include a placeholder:

```html

<li>No posts yet. Check back soon!</li>

```

– Future posts will be `posts/post-YYYY-MM-DD-title.html` and linked here.

3. \*\*Navigation\*\*

– Include a “Back to Home” link or nav bar linking to `index.html` and `blog.html`.

\*\*Design Guidelines for `blog.html`:\*\*

• Match header/footer styles and accent colors from `index.html`.

• Center content in a container (`max-width: 800px; margin: 0 auto; padding: 20px`).

• Each post link (when added) appears as:

```html

<li>

<a href="posts/post-2024-07-05-how-to-avoid-drain-clogs.html">How to Avoid Drain Clogs This Summer</a>

<span style="color: #666;">(July 5, 2024)</span>

</li>

### **3. Create /posts/ Folder**

* Simply generate an empty directory named posts/.
* No files inside initially. Future blog posts go here (name format: post-YYYY-MM-DD-title.html).

posts/

└── (empty)

---

### 2.5 `README.md`

Must include exactly these sections (copy verbatim):

---

# 5MM Website Analyzer & Site Builder

### 1. Forking & Setup

1. Click “Fork” in the top-right corner of this Replit project.

2. In your fork, click the “Files” tab → create a file named `.env`.

3. Paste your OpenAI API key into `.env` as: OPENAI\_API\_KEY=sk-…

4. (Optional) If login is enabled, open `authorized\_users.py` and add your email to `AUTHORIZED\_EMAILS`.

### 2. Running the Analyzer

1. Click “Run” (green ▶️ button).

2. A “Web View” pane will open on the right (Streamlit interface).

3. (If login is enabled) Enter your authorized email → click “Login.”

4. On the “Enter Your OpenAI API Key” screen, paste your key → click “Continue.”

5. In the main interface:

- Paste any local business website URL (e.g., `exampleplumbing.com`).

- Click \*\*“Analyze Website.”\*\*

6. Wait for the analysis spinner. Once finished, you’ll see:

- \*\*Score (1–10)\*\* with color (red/yellow/green).

- \*\*Plain-English Report\*\* listing each deficiency.

- \*\*“Is This a Good Lead?”\*\* (Yes/No + rationale).

- \*\*Replit Prompt\*\* (copy-ready to generate a fresh site).

- \*\*Outreach Email\*\* and \*\*Outreach DM\*\* templates (click “Copy” to copy text).

7. To export:

- Click \*\*“Copy All”\*\* to copy everything to clipboard.

- Click \*\*“Download PDF”\*\* to download a PDF of the report (optional).

- Click \*\*“Reset”\*\* to clear and analyze another site.

### 3. Using the Replit Prompt

1. Copy the \*\*entire contents\*\* of the “Replit Prompt” box.

2. Create a \*\*new Replit AI Site Builder\*\* or ChatGPT “create a site” session.

3. Paste the prompt exactly.

4. The AI will generate:

- \*\*index.html\*\* (homepage demo)

- \*\*blog.html\*\* (blog index page)

- An empty `/posts/` folder

- A shared `styles.css` file

5. Review, edit any content if needed (e.g., phone # or service wording).

6. Click \*\*Deploy → Static Site\*\* in Replit to go live at `clientname.replit.app`.

### 4. Folder Structure After Site Build

/

├── index.html

├── blog.html

├── posts/ # initially empty

└── styles.css

- Future OTO3 blog posts will automatically write new HTML files into `/posts/` (e.g. `post-2024-07-06-topic.html`) and update `blog.html` with new links—without touching `index.html`.

---

## 3. Analyzer Backend Logic (Python)

All code must live in \*\*`analyzer\_utils.py`\*\*. The Replit agent should create this file exactly with the functions below. Do \*\*not\*\* use any additional or unprescribed dependencies.

```python

# analyzer\_utils.py

import re

import time

import requests

from bs4 import BeautifulSoup

import openai

# ---------- FETCH & TIMING ----------

def fetch\_html(url: str, timeout: int = 10):

"""

Returns (soup, load\_time\_seconds).

Ensures `https://` is prepended if missing.

"""

if not url.lower().startswith("http"):

url = "https://" + url

start = time.time()

response = requests.get(url, timeout=timeout)

load\_time = time.time() - start

response.raise\_for\_status()

soup = BeautifulSoup(response.text, "html.parser")

return soup, load\_time

# ---------- RULE-BASED CHECKS ----------

def check\_ssl(url: str):

if url.lower().startswith("https://"):

return True, ""

return False, "No SSL (site not secure)."

def check\_mobile\_meta(soup: BeautifulSoup):

if soup.find("meta", {"name": "viewport"}):

return True, ""

return False, "Missing mobile viewport meta tag (not mobile-friendly)."

def check\_page\_speed(load\_time: float, threshold: float = 4.0):

if load\_time <= threshold:

return True, ""

return False, f"Page load time {load\_time:.1f}s (slower than {threshold}s)."

def check\_company\_name\_or\_title(soup: BeautifulSoup):

title = soup.title.string if soup.title else ""

if title and ("home" not in title.lower() and "index" not in title.lower() and len(title.strip()) > 5):

return True, ""

return False, "Missing clear business name in <title> or header."

def check\_contact\_cta(soup: BeautifulSoup):

# Look for tel: or mailto: or any “contact” link

if soup.find("a", href=lambda x: x and ("tel:" in x or "mailto:" in x)):

return True, ""

if soup.find("a", href=lambda x: x and "contact" in x.lower()):

return True, ""

return False, "No clear Call-to-Action (phone or contact link)."

def check\_hero\_section(soup: BeautifulSoup):

hero = soup.find(["h1", "h2"])

if not hero:

return False, "No H1/H2 headline found in hero area."

if soup.find("button") or soup.find("a", href=lambda x: x and ("call" in x.lower() or "book" in x.lower())):

return True, ""

return False, "No hero CTA/button found."

def check\_navigation(soup: BeautifulSoup):

nav = soup.find("nav")

if nav:

links = nav.find\_all("a")

if len(links) >= 3:

return True, ""

return False, "Navigation menu missing or fewer than 3 links."

def check\_outdated\_html(soup: BeautifulSoup):

outdated\_tags = ["center", "font", "marquee", "frameset"]

found = [tag for tag in outdated\_tags if soup.find(tag)]

if found:

return False, f"Found outdated HTML tags: {', '.join(found)}."

return True, ""

def check\_images(soup: BeautifulSoup):

imgs = soup.find\_all("img")

if not imgs:

return False, "No images found (poor visual appeal)."

broken = sum(1 for img in imgs if not img.get("src"))

if broken > 0:

return False, f"{broken} broken image(s) detected."

return True, ""

def check\_footer\_date(soup: BeautifulSoup):

text = soup.get\_text()

match = re.search(r"©\s\*(\d{4})", text)

if match:

year = int(match.group(1))

if year < 2018:

return False, f"Copyright footer year is {year} (outdated)."

return True, ""

return False, "No copyright date found."

def check\_trust\_signals(soup: BeautifulSoup):

if soup.find(string=lambda t: t and ("review" in t.lower() or "testimonial" in t.lower())):

return True, ""

if soup.find("img", alt=lambda a: a and ("bbb" in a.lower() or "google" in a.lower())):

return True, ""

return False, "No visible trust signals (reviews, badges)."

def check\_about\_section(soup: BeautifulSoup):

if soup.find(string=lambda t: t and "about" in t.lower()):

return True, ""

return False, "No About/Team section found."

def check\_local\_seo\_signals(soup: BeautifulSoup):

if soup.find("iframe", src=lambda x: x and "google.com/maps" in x):

return True, ""

if soup.find(string=lambda t: t and re.search(r"\b\d{5}(?:-\d{4})?\b", t)):

return True, ""

return False, "No location/address information detected."

def compute\_score\_and\_reasons(url: str):

"""

Runs all checks and returns (final\_score:int, reasons:List[str]).

Deduct points per failed check; clamp between 1–10.

"""

soup, load\_time = fetch\_html(url)

reasons = []

score = 10

checks = [

(check\_ssl, 2),

(check\_mobile\_meta, 2),

(lambda s: check\_page\_speed(load\_time), 1),

(check\_company\_name\_or\_title, 1),

(check\_contact\_cta, 1),

(check\_hero\_section, 1),

(check\_navigation, 1),

(check\_outdated\_html, 1),

(check\_images, 1),

(check\_footer\_date, 1),

(check\_trust\_signals, 1),

(check\_about\_section, 1),

(check\_local\_seo\_signals, 1)

]

for check\_func, deduction in checks:

ok, reason = check\_func(soup if check\_func != check\_page\_speed else load\_time)

if not ok:

score -= deduction

reasons.append(reason)

final = max(1, min(10, score))

return final, reasons

# ---------- GPT INTEGRATION ----------

ANALYSIS\_PROMPT = """

You are a professional web auditor. A local business website produced these findings:

{rule\_summary}

The rule-based “raw score” is {score}/10.

Please:

1. Tell me if this is a good lead (Yes/No).

2. Provide a brief rationale (1–2 sentences).

3. Confirm or adjust the score out of 10.

"""

OUTREACH\_PROMPT = """

Write both:

1) A friendly outreach email, and

2) A short outreach DM,

to propose a no-strings-attached demo site to this type of business.

Business type: {business\_type}

Location: {location}

Key web issues: {issues\_summary}

Offer: “I built you a free demo site. Check it out here: [demo link]. No strings attached.”

Output format:

---EMAIL---

Subject: …

Body: …

---DM---

…

"""

def analyze\_with\_gpt(rule\_summary: str, score: int):

prompt = ANALYSIS\_PROMPT.format(rule\_summary=rule\_summary, score=score)

response = openai.ChatCompletion.create(

model="gpt-4o-mini",

messages=[{"role": "system", "content": prompt}],

max\_tokens=200

)

return response.choices[0].message["content"].strip()

def generate\_outreach\_messages(business\_type: str, location: str, issues\_summary: str):

prompt = OUTREACH\_PROMPT.format(

business\_type=business\_type,

location=location,

issues\_summary=issues\_summary

)

response = openai.ChatCompletion.create(

model="gpt-4o-mini",

messages=[{"role": "system", "content": prompt}],

max\_tokens=300

)

content = response.choices[0].message["content"].strip()

# Split into EMAIL & DM

if "---EMAIL---" in content and "---DM---" in content:

email\_part, dm\_part = content.split("---DM---", 1)

email\_text = email\_part.replace("---EMAIL---", "").strip()

dm\_text = dm\_part.strip()

else:

email\_text = content

dm\_text = ""

return email\_text, dm\_text

## **4. Streamlit Application (main.py)**

Below is the exact code structure. Do **not** modify any section headings or logic flow unexpectedly. Every comment and UI element must appear as shown so the user experience matches the wireframe exactly.

# main.py

import streamlit as st

import openai

from analyzer\_utils import compute\_score\_and\_reasons, analyze\_with\_gpt, generate\_outreach\_messages

# ---------- CONFIGURATION ----------

USE\_LOGIN = False # Set to True only if authorized\_users.py is furnished

# ---------- LOGIN SCREEN (Optional) ----------

if USE\_LOGIN:

if "authorized" not in st.session\_state or not st.session\_state["authorized"]:

st.title("🔐 Authorized Access Only")

email = st.text\_input("Enter your email address")

if st.button("Login"):

from authorized\_users import AUTHORIZED\_EMAILS

if email.strip().lower() in [e.lower() for e in AUTHORIZED\_EMAILS]:

st.session\_state["authorized"] = True

st.experimental\_rerun()

else:

st.error("Access denied. Please contact support.")

st.stop()

# ---------- API KEY SCREEN ----------

if "api\_key" not in st.session\_state:

st.title("🔑 Enter Your OpenAI API Key")

api\_key\_input = st.text\_input("Paste your OpenAI API key", type="password")

if st.button("Continue"):

key = api\_key\_input.strip()

if key:

st.session\_state["api\_key"] = key

openai.api\_key = key

st.experimental\_rerun()

else:

st.error("Please enter a valid API key.")

st.stop()

# ---------- MAIN INTERFACE ----------

st.title("🔍 5MM Website Analyzer Tool")

url\_input = st.text\_input("Website URL", placeholder="e.g. exampleplumbing.com")

if st.button("Analyze Website") and url\_input.strip():

try:

# 1. Fetch & Score

final\_score, reasons = compute\_score\_and\_reasons(url\_input.strip())

rule\_summary = "\n".join(f"- {r}" for r in reasons)

# 2. GPT Analysis

analysis\_text = analyze\_with\_gpt(rule\_summary, final\_score)

verdict\_line = analysis\_text.splitlines()[0].strip()

# 3. Outreach Messages (use placeholders for business\_type & location if not known)

business\_type = "Local Business"

location = "City, State"

outreach\_email, outreach\_dm = generate\_outreach\_messages(

business\_type, location, rule\_summary

)

# 4. Replit Prompt (load the template and fill placeholders)

with open("replit\_prompt\_template.txt", "r") as f:

prompt\_template = f.read()

replit\_prompt = prompt\_template.format(

domain=url\_input.strip(),

business\_type=business\_type,

location=location

)

# 5. Store in session\_state

st.session\_state["score"] = final\_score

st.session\_state["analysis\_text"] = analysis\_text

st.session\_state["verdict"] = verdict\_line

st.session\_state["replit\_prompt"] = replit\_prompt

st.session\_state["outreach\_email"] = outreach\_email

st.session\_state["outreach\_dm"] = outreach\_dm

except Exception as e:

st.error(f"Error during analysis: {e}")

st.stop()

# ---------- DISPLAY RESULTS (If Available) ----------

if "score" in st.session\_state:

domain\_display = url\_input.strip()

# Score & Color

score = st.session\_state["score"]

color = "red" if score <= 3 else "orange" if score <= 6 else "green"

st.markdown(f"<h1 style='color:{color};'>Score: {score}/10</h1>", unsafe\_allow\_html=True)

# Plain-English Report

st.subheader("Plain-English Report")

st.text\_area("", st.session\_state["analysis\_text"], height=150)

# Good Lead Verdict

st.subheader("Is This a Good Lead?")

st.markdown(f"\*\*{st.session\_state['verdict']}\*\*")

# Replit Prompt

st.subheader("Replit Prompt")

st.text\_area("", st.session\_state["replit\_prompt"], height=200)

# Outreach Email

st.subheader("Outreach Email")

st.text\_area("", st.session\_state["outreach\_email"], height=150)

if st.button("Copy Email"):

st.experimental\_set\_clipboard(st.session\_state["outreach\_email"])

st.success("Email copied to clipboard")

# Outreach DM

st.subheader("Outreach DM")

st.text\_area("", st.session\_state["outreach\_dm"], height=100)

if st.button("Copy DM"):

st.experimental\_set\_clipboard(st.session\_state["outreach\_dm"])

st.success("DM text copied to clipboard")

# Export / Reset Buttons

col1, col2, col3 = st.columns(3)

with col1:

if st.button("Copy All"):

all\_text = (

f"Score: {score}/10\n\n"

f"Report:\n{st.session\_state['analysis\_text']}\n\n"

f"Verdict: {st.session\_state['verdict']}\n\n"

f"Replit Prompt:\n{st.session\_state['replit\_prompt']}\n\n"

f"Email:\n{st.session\_state['outreach\_email']}\n\n"

f"DM:\n{st.session\_state['outreach\_dm']}"

)

st.experimental\_set\_clipboard(all\_text)

st.success("All content copied to clipboard")

with col2:

try:

from pdf\_generator import generate\_pdf

pdf\_bytes = generate\_pdf(

domain\_display,

score,

st.session\_state["analysis\_text"],

st.session\_state["verdict"],

st.session\_state["replit\_prompt"],

st.session\_state["outreach\_email"],

st.session\_state["outreach\_dm"]

)

st.download\_button("Download PDF", data=pdf\_bytes, file\_name=f"{domain\_display}-analysis.pdf", mime="application/pdf")

except Exception:

st.error("PDF export is not available.")

with col3:

if st.button("Reset"):

for key in ["score", "analysis\_text", "verdict", "replit\_prompt", "outreach\_email", "outreach\_dm"]:

del st.session\_state[key]

st.experimental\_rerun()

## **5. Additional Notes & Strict Requirements**

1. **No Extra Dependencies:**
   * Only install packages listed in requirements.txt.
   * Don’t add any new Python libraries or JavaScript files.
2. **Exact UI Layout:**
   * Do not rearrange or rename any Streamlit headers (st.title, st.subheader, etc.).
   * Buttons (“Analyze Website,” “Copy Email,” “Copy DM,” “Copy All,” “Download PDF,” “Reset”) must appear in the order shown.
3. **Replit Prompt Must Match Exactly:**
   * The contents of replit\_prompt\_template.txt cannot be changed except for replacing the placeholders {domain}, {business\_type}, and {location} at runtime.
   * Do not add or remove any sections or bullet points in that template.
4. **Folder Structure Enforcement:**
   * After site generation (via AI prompt), the folder structure must be:

index.html

blog.html

posts/ ← empty

styles.css ← if included

* + No other files in root. This ensures OTO3’s auto-poster can operate safely.

1. **Session‐Only API Keys & No Logging:**
   * Do not write the user’s OpenAI API key to disk.
   * Store it only in st.session\_state["api\_key"].
   * Do not print or log the API key.
2. **Login Layer (Optional):**
   * If USE\_LOGIN = True, the app must enforce login before any other UI.
   * Unauthorized users see only the login screen and cannot proceed.
   * If USE\_LOGIN = False, skip that screen entirely.
3. **Error Handling:**
   * If any exception occurs during analysis (network failure, invalid URL, GPT error), show a user‐friendly st.error("Error during analysis: …") and stop further execution.
   * Under no circumstances should a Python traceback appear to the end user.

## **6. Summary**

* **Reader of this spec:** You are to create a forkable Replit project containing exactly the files and code outlined above.
* **Key Goals:**
  1. Build a robust **Website Analyzer** with rule-based checks + GPT-powered verdict/outreach.
  2. Ensure the “Replit Prompt” generates a **two‐page site** (index.html + blog.html + empty /posts/ folder).
  3. Match the wireframe and UI flow exactly—no extra elements or changes.
  4. Provide a clear README.md so any user can fork, paste their OpenAI key, and start analyzing immediately.