HTML AND JS ANALYZER UNMINIFY & RETRO ENGINEERING

IDEO-LAB

JANUARY 2026

VERSION 1.0

OBJECTIFS & PERIMETRES

AGENDA OBJECTIFS & PERIMETRES

- 1) Objectif produit
- 2) Cas d’usage (ciblés)
- 3) Entrées / sorties
- 4) Non-objectifs (v1)
- 5) Contraintes techniques

1) Objectif produit

Créer une app Django `unminify/` capable de :

1. **Unminify** CSS / HTML / JS (formatage + normalisation)
2. **Dé-obfuscation légère** (patterns courants, sans "casser" le code)
3. **Rétro-engineering JS** : extraire structure & flux
4. Produire des **artefacts lisibles** : graphes, index, rapports, "call flow", dépendances

2) Cas d'usage (ciblés)

- Analyser un bundle minifié (site, SaaS, script)
- Comprendre "qui appelle quoi", quelles classes, quels modules, quelles routes
- Repérer : trackers, endpoints, événements, injections DOM, i18n, etc.
- Faire de la **due diligence** (sécurité, conformité, perf)
- Aider à la maintenance d'un legacy bundle interne

3) Entrées / sorties

Entrées

- fichiers `.js`, `.css`, `.html`
- bundles + sourcemaps (`.map`)
- option : URLs (fetch) + téléchargement en sandbox

Sorties

- `*.pretty.js` / `*.pretty.css` / `*.pretty.html`
- `report.html` (lisible, navigable)
- index symboles (fonctions/classes/vars)
- graphe d'appels (call graph)
- graphe d'imports/dépendances
- timeline "startup flow" (entrypoints + listeners)
- export JSON (pour scripts / API)

4) Non-objectifs (v1)

- Déobfuscation "agressive" type malware reversing complet
- Reconstruction parfaite des noms originaux sans sourcemap
- Exécution en sandbox JS complète (on fera plus tard)

5) Contraintes techniques

- **100% offline** possible (pas besoin de web)
- Déterministe, reproductible
- Traitement asynchrone (Celery) + progression persistante
- UI Ideo-Lab : dashboard + modals + actions (copy/print)
- Multi-projets : "Workspace / Jobs / Artifacts"
- Stockage : filesystem + DB métadonnées

CDCF V 1.0

AGENDA

- 1) Objectif produit
- 2) Périmètre v1.0 (MVP “utile”)
- 3) Hors périmètre v1.0 (v2+)
- 4) Cibles utilisateurs
- 5) Contraintes non-fonctionnelles
- 6) Architecture technique (cible)
- 7) Livrables v1.0

CDCF v1.0 — Cahier des charges fonctionnel

1) Objectif produit

Construire un outil web (Django) permettant de :

- **Importer** des fichiers minifiés (ou obfusqués "léger") : `.js`, `.css`, `.html`
- **Unminify / Beautify** avec réglages reproductibles
- **Produire des artefacts** exploitables : fichiers reformatés + rapports d'analyse
- **Rétro-engineer JS** : compréhension structurelle, points d'entrée, endpoints réseau, events, dépendances, modules, etc.
- **Comparer** (optionnel v1.1) : original vs unminified, versions, deltas

2) Périmètre v1.0 (MVP “utile”)

Entrées

- Upload fichier(s) (drag & drop)
- Coller du code (textarea) + bouton “Analyser”
- URL distante (optionnel v1.1 : fetch sécurisé)

Sorties

- Code “beautifié” (prévisualisation + téléchargement)
- Rapport JSON + rapport HTML lisible
- Index “projet” : historique des analyses, tags, notes notes

Analyse JS (v1.0)

- Détection du style : IIFE, UMD, CJS, ESM, Webpack bundle, Rollup, Vite, etc.
- Extraction :
  - endpoints (`fetch`, `XMLHttpRequest`, axios), URLs, routes
  - clés sensibles **probables** (API keys hardcodées, tokens, secrets) → signalement (pas de fuite)
  - événements DOM, listeners, timers
  - globals exportés / fenêtre (`window.*`)
  - dépendances / modules / imports / requires si présents
- Graphe :
  - liste fonctions top-level
  - callgraph **approximatif** (statique)
  - liste des “entry points” (DOMContentLoaded, onload, handlers, exports)

CSS/HTML (v1.0)

- Beautify / format + rapport simple :
  - CSS : sélecteurs, media queries, variables, taille, spécificité approximative
  - HTML : hiérarchie DOM, scripts/styles externes, inline scripts

3) Hors périmètre v1.0 (v2+)

- Déobfuscation “hard” (control-flow flattening, string encryption avancée)
- Exécution / instrumentation runtime dans navigateur (devtools-like)
- Désassemblage wasm / sourcemaps auto-fetch (sourcemaps v1.1 possible)
- IA de “renommage intelligent” complet (v2 : LLM assist)

4) Cibles utilisateurs

- Toi (Ideo-Lab) en premier client
- Devs qui auditent un bundle tiers
- Security/ops (audit endpoints, tracking, code suspect)
- Debug front sur code minifié en prod (sans sourcemap)

5) Contraintes non-fonctionnelles

- **Sécurité** : sandbox parsing, taille max upload, pas d’exécution JS serveur
- **Performance** : traitement async via Celery (gros bundles)
- **Traçabilité** : versions d’analyses, hash des fichiers, reproductibilité des options
- **UX Ideo-Lab** : page grid de cards + modals (comme tes guides)
- **Export** : ZIP téléchargeable (code unminified + rapport + metadata)

6) Architecture technique (cible)

- Django app `unminify/`
- Workers async (Celery + Redis) pour :
  - unminify gros fichiers
  - analyse AST JS
  - génération rapports / graph
- Stockage :
  - DB : projets, fichiers, jobs, résultats, métriques
  - FS (media/) : fichiers bruts + fichiers générés
- Librairies pressenties (côté serveur) :
  - JS parse/AST : *tree-sitter* ou *esprima/acorn* via wrapper, ou Node sandbox (v1.1)
  - Beautify : jsbeautifier / postcss / html-beautify (à trancher à l'implémentation)

7) Livrables v1.0

- App Django complète : models/admin/views/urls/templates/static
- Dashboard + pages :
  - New Analysis
  - Project History
  - Result Viewer (code + rapport + downloads)
- Commandes :
  - `manage.py unminify_reindex` (optionnel)
- API interne :
  - endpoints JSON pour polling job/progress

AF v1.0 — Analyse fonctionnelle

A) Parcours principal (MVP)

1. **Créer une analyse**
   - L'utilisateur arrive sur `/unminify/`
   - Card "New Analysis" → modal :
     - Upload fichier(s) OU coller du code
     - Type auto-detect + override
     - Options (indent, quotes, wrap line length, preserve comments…)
     - Bouton **Run**
2. **Job async**
   - Création job + affichage progress (polling)
   - États : queued → running → done / failed
3. **Résultat**
   - Viewer avec onglets :
     - **Unminified Code**
     - **Report (HTML)**
     - **JSON**
     - **Findings** (endpoints, globals, events, suspicious strings)
     - **Callgraph (list v1.0)** (v1.1 : graph visuel)
   - Boutons : Download ZIP / Copy / Print

B) Écrans (format Ideo-Lab)

- **Dashboard** (grid kcards)
  - New Analysis
  - Recent Jobs
  - Saved Projects
  - Stats (nb fichiers, tailles, temps)
- **Job Detail**
  - barre de progression persistante
  - logs “humains” (5–20 lignes max + option voir tout)
- **Result Viewer**
  - code viewer (monospace, line numbers, search)
  - findings panel (badges + copy)
  - downloads

C) Règles métier

- Un fichier = hash SHA256 → dédup possible (stockage)
- Options d'unminify versionnées avec le résultat (reproductible)
- Limites :
  - taille fichier (ex 10–25MB v1.0)
  - nombre fichiers par batch (ex 20)
- Confidentialité :
  - par défaut privé (superuser-only option de partage public)

D) Mesures / KPI internes

- durée analyse
- taille entrée/sortie
- nb endpoints trouvés
- nb listeners / globals / exports
- score "complexité" (heuristique)

AF v1.0 — Analyse Détaillée

A) Modules pipeline (cœur)

1. **Ingest**
   - upload multi-fichiers
   - détection type (js/css/html/map)
   - création “Job” + “Artifacts”
2. **Format / Beautify**
   - JS : prettier / js-beautify
   - CSS : prettier / postcss + formatter
   - HTML : prettier
3. **Sourcemap resolver (gros gain)**
   - si `.map` dispo : remapper
   - remonter noms, fichiers, lignes
   - reconstruire arborescence “sources/”
4. **JS Parse & Index**
   - parse en AST (Babel / Acorn)
   - extraire :
     - fonctions (decl/expr/arrow)
     - classes + méthodes
     - variables exported/imported
     - appels (CallExpression)
     - events (addEventListener, on*)
     - accès réseau (fetch/XHR/WebSocket)
     - DOM mutations (querySelector, innerHTML, appendChild)
     - timers (setTimeout/setInterval)
   - construire un **Symbol Table** + cross refs

5. **Call Graph & Flow**
    - graphe “who calls who”
    - entrypoints :
        - IIFE, top-level calls
        - DOMContentLoaded
        - listeners
        - framework hooks (React/Vue/Angular heuristics)
    - “Functional flow” : chemins probables
6. **Rename heuristics (option v1.1)**
    - renommer `a,b,c` -> `fn_12`, `class_5`, `mod_net`
    - basé sur usage (fetch -> `net_*`, dom -> `dom_*`)
7. **Report builder**
    - HTML report :
        - onglets : Overview / Symbols / Calls / Network / DOM / Events / Files
    - export JSON

UI UnMinify - Ideo-Lab

UI (Ideo-Lab) — ce que tu vas aimer

- Page dashboard `unminify/` :
  - cards “New Job”, “Recent Jobs”, “Artifacts”
  - chaque job ouvre une modal “Job details”
- “Report Viewer” :
  - navigation gauche (files)
  - panneau droit (symbols/calls)
- Boutons :
  - Copy (liens, snippets)
  - Print (report)
  - Download (zip artifacts)
- Progress bar **persistante** (comme ton point Weglot) :
  - polling `/api/jobs/<id>/status`
  - Celery updates en DB

DatModel Prévisionnel

Modèle de données minimal (v1)

- `UnminifyProject` (optionnel)
- `UnminifyJob` (status, progress, started/ended, input_meta)
- `UnminifyFile` (original + type + hash)
- `UnminifyArtifact` (pretty files, report, json exports)
- `UnminifySymbol` (name, kind, file, loc, signature)
- `UnminifyEdge` (caller -> callee, type, weight)
- `UnminifyFinding` (network endpoints, dom sinks, etc.)

Tech stack recommandée (concrète)

- Backend Django + Celery + Redis
- Analyse JS via Node tooling (fiable) :
  - `prettier`
  - `@babel/parser` + traverse
  - `source-map` (resolver)
- Appel depuis Django :
  - subprocess contrôlé + timeouts + sandbox dir
- Stockage artifacts :
  - `/var/lib/ideo-lab/unminify/jobs/<uuid>/...`

Architecture du Projet

```
unminify/
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tasks.py
  urls.py
  views.py
  templates/
    unminify/
      dashboard.html
      job_detail.html
      report_view.html
  static/
    unminify/
      unminify.js

unminify_engine/
  package.json
  run.js
  lib/
    io.js
    beautify.js
    analyze.js
    report.js
```

INSTALLATION

UnMinfify Engine & Node