

RAID — Documentation technique

Procédure — Évolution du script FlexHistoApi en outil de stats et site web

Développeur	COUR Cyrille
Organisation	Projet personnel — team Ronin
Formation	BTS SIO SLAM — Alternance 2025/2027
Stack	Python 3.13 / PHP 8.4 / JavaScript (modules ES)
Date	15/10/2026
Version	2.0

Sommaire

Sommaire	2
1. Contexte	3
1.1 De FlexHistoApi à RAID	3
1.2 Périmètre.....	3
2. Environnement.....	4
2.1 Prérequis	4
2.2 Dépendances.....	4
2.3 Installation	4
3. Procédure	5
3.1 Étape 1 — Configurer la clé API et le bloc CONFIG	5
<i>Détail technique</i>	5
3.2 Étape 2 — Récupérer les games et alimenter le cache	6
<i>Détail technique</i>	6
3.3 Étape 3 — Extraire la phase de lane à 15 minutes.....	6
<i>Détail technique</i>	7
3.4 Étape 4 — Détecter les ganks et les invades	7
<i>Détail technique</i>	7
3.5 Étape 5 — Noter la game et désigner MVP / SVP	8
<i>Détail technique</i>	8
3.6 Étape 6 — Générer l'Excel et saisir les WR de matchups.....	8
<i>Détail technique</i>	8
3.7 Étape 7 — Exporter les données et servir le site.....	10
<i>Détail technique</i>	10
3.8 Étape 8 — Vérifier avec les tests	10
<i>Détail technique</i>	10
4. Structure / Données.....	11
4.1 Clés de data/site.json	11
4.2 Arborescence utile	11
5. Déploiement / CI-CD	12
6. Points de vigilance	12

1. Contexte

RAID (Ronin Analyse d'Information Détaillé) récupère l'historique des games de la team Ronin via l'API Riot et le transforme en statistiques : un fichier Excel d'analyse et un site web façon DPM.lol, centrés sur la phase de lane à 15 minutes.

Le projet est l'évolution de FlexHistoApi, un petit script Python qui comparait nos lanes en Flex 5v5 (EUW) à celles de nos adversaires directs et générait un Excel. Le script a grossi par étapes : ganks, invades, note de game, MVP / SVP, forme récente, puis un site pour consulter les stats sans ouvrir Excel.

1.1 De FlexHistoApi à RAID

Domaine	FlexHistoApi (origine)	RAID (aujourd'hui)
Files	Flex 5v5 (EUW)	Flex, Clash, Team 5v5, Solo/Duo + comptes secondaires
Lane à 15 min	Écarts face à l'adversaire direct	CS / gold / XP / vision early / KDA, verdict de lane (3 critères sur 5), avances max, solo kills, plaques, dives
Jungle et roams	—	Ganks, ganks reçus, invades, first clear
Évaluation	WR de matchup saisi à la main (DPM)	Note sur 10 façon DPM.lol, MVP / SVP, section Forme (5 dernières games)
Équipe	—	Onglets dédiés aux games jouées à 5
Sorties	Fichier Excel	stats_RONIN.xlsx + data/site.json + site PHP
Qualité	—	353 tests pytest, 108 PHPUnit, 50 Vitest

1.2 Périmètre

- Outil interne des 5 joueurs de Ronin, réglé sur nos comptes, nos postes et nos files : ce n'est pas un site public.
- Le calcul (main.py) et l'affichage (site PHP) sont séparés : le site ne fait que lire un JSON déjà calculé.
- Le site tourne en local sur Laragon (virtual host raid.test).

Note — Cette procédure s'adresse à quiconque reprend le dépôt : elle explique comment installer RAID, comment le traitement s'enchaîne de l'API Riot jusqu'au site, et où intervenir pour faire évoluer un calcul.

2. Environnement

2.1 Prérequis

Outil	Version	Rôle
Python	≥ 3.8 (testé en 3.13.7)	Tout le calcul : API Riot, cache, stats, Excel, export du site
PHP	>= 8.2 (testé en 8.4.24)	Le site : routage et rendu HTML depuis data/site.json
Composer	2.10.3	Autoload PSR-4 du site et PHPUnit
Node.js / npm	24.20.0 / 11.19.0	Uniquement pour les tests JS (Vitest)
Apache (Laragon)	—	Sert public/ et réécrit les URL vers index.php
Clé API Riot	Clé de dev	developer.riotgames.com — expire au bout de 24 h

2.2 Dépendances

Paquet	Version	Rôle
requests (pip)	2.34.2	Appels API Riot et Data Dragon (session, retries)
openpyxl (pip)	3.1.5	Écriture de l'Excel (formules, mises en forme) et relecture des WR saisis
python-dotenv (pip)	1.2.1	Lecture de RIOT_API_KEY dans .env
Chart.js (CDN jsDelivr)	4.5.0	Graphes des pages joueur et équipe
pytest / phpunit / vitest	9.1.1 / 13.4.2 / 5.0.4	Tests Python, PHP, JS

Note — Aucune dépendance PHP de production, pas de requirements.txt côté Python (paquets installés à la main), pas de build front : le JS est servi tel quel en modules ES natifs.

2.3 Installation

```
$ git clone <url-du-depot> RAID
$ cd RAID
$ pip install requests openpyxl python-dotenv
$ cp .env.example .env           # puis RIOT_API_KEY=RGAPI-...
$ composer install               # autoload PHP (+ PHPUnit)
$ npm install                    # uniquement pour les tests JS
```

3. Procédure

Le traitement se fait en deux temps : un script de calcul lancé à la main (main.py), qui produit l'Excel et un JSON, puis un site PHP rendu côté serveur qui n'affiche que ce JSON.

```
python main.py
├─ main()
│   ├── charger_noms_champions() Data Dragon
│   ├── puuid_de() / ids_matches() API Riot : comptes, games
│   ├── match_et_timeline()      cache, sinon API (429 géré)
│   └── extraire_game()          lane, ganks, invades, stats
├─ construire_excel() → stats_RONIN.xlsx
└─ exporter_web()     → data/site.json
```

```
Navigateur → public/.htaccess → public/index.php
→ App::reponse() → Routeur::resoudre()
→ Site (lecture du JSON) → Vue::rendre()
```

3.1 Étape 1 — Configurer la clé API et le bloc CONFIG

Détail technique

La clé Riot se place dans .env (ignoré par Git) et est chargée au démarrage par python-dotenv. Une variable d'environnement RIOT_API_KEY déjà définie est prioritaire sur le .env.

```
# PowerShell
$env:RIOT_API_KEY = "RGAPI-..."
# bash
export RIOT_API_KEY="RGAPI-..."
```

Tous les réglages sont regroupés dans le bloc CONFIG en haut de main.py :

Variable	Rôle	Valeur
JOUEURS	Riot ID complets (pseudo#tag)	les 5 membres
DATE_DEBUT	Date de début de récupération	2026-09-15
FILES	Files récupérées {id: nom}	Flex, Clash, Team 5v5, Solo/Duo
MINUTE	Minute de comparaison de la lane	15
POSTES	Poste conservé par joueur	un poste par joueur
COMPTESECONDAIRES	Comptes ajoutés au compte principal	—
GANK_DEBUT / FIN / MARGE	Fenêtre et marge de détection des ganks	3 / 15 / 60 s
INVADE_MARGE / ECART	Recul de la ligne des tours / écart max entre actions	700 unités / 60 s
SEUIL_THROW	Avance (ou retard) d'équipe pour un throw (ou comeback)	4000 gold
FORME_GAMES	Games prises en compte pour la Forme	5
FICHIER_WEB	Données du site (None = pas d'export)	data/site.json

Attention — POSTES et COMPTESECONDAIRES ne doivent citer que des comptes de JOUEURS : sinon le script s'arrête volontairement, pour qu'une faute de frappe ne soit pas ignorée en silence.

```
# ===== CONFIG =====
def cle_api(chemin_env=None):
    """Clé lue dans .env (jamais commité). override=True : le .env passe devant une RIOT_API_KEY déjà présente dans
    l'environnement (terminal VS Code ouvert avec l'ancienne clé, $env:RIOT_API_KEY...), sinon une clé changée dans
    .env serait ignorée."""
    load_dotenv(chemin_env, override=True)
    return os.environ.get("RIOT_API_KEY", "RGAPI-xxxx")

API_KEY = cle_api()
JOUEURS = ["Abanox#TOJO", "Mel En Chon 2027#RONIN", "Bard Est Là 2027#G2WIN", "Liegas#Liego", "LE SSERAFINE#LUNA"] # Riot ID complets (pseudo#tag)
DATE_DEBUT = "2026-09-15" # première flex à 5
FILES = {
    440: "Flex",
    700: "Clash",
    710: "Team 5v5", | Cyrille, la semaine dernière + ajout timer jg & poste des joueurs ...
    420: "Solo/Duo",
}
# 440 = flex, 420 = solo/duo, 700 = clash, 710 = team 5v5
REGIONAL = "europe" # routage régional pour EUM
MINUTE = 15
FICHIER_SORTIE = "stats_RONIN.xlsx" # Fichier Excel final
FICHIER_WEB = "data/site.json" # données du site web (lues par public/) ; None = pas d'export
DOSSIER_CACHE = "cache" # un fichier compressé par game, partagé entre tous les modes
ANCIEN_CACHE = "cache_matches.json" # migré automatiquement vers DOSSIER_CACHE au premier lancement
JOUEURS_JUNGLE = ["Mel En Chon 2027#RONIN"] # comptes pour lesquels on affiche le timer du first clear
NOM_EQUIPE = "Ronin" # onglets Games/Champions/Global des games jouées à 5 ensemble
# Ne garder que les games jouées à ce poste (TOP, JUNGLE, MIDDLE, BOTTOM, UTILITY). Compte absent = toutes ses games.
POSTES = {
    "Abanox#TOJO": "TOP",
    "Mel En Chon 2027#RONIN": "JUNGLE",
    "Bard Est Là 2027#G2WIN": "MIDDLE",
    "Liegas#Liego": "BOTTOM",
    "LE SSERAFINE#LUNA": "UTILITY"
}
# Comptes secondaires : leurs games sont ajoutées à celles du compte principal (mêmes onglets, même filtre de poste).
COMPTES_SECONDAIRES = {
    "Bard Est Là 2027#G2WIN": ["WhiteDrou#EUM"],
    "Abanox#TOJO": ["Abanox#YEET", "João juventino#1897", "Zequid#EUM", "AbanoxIS#JUVE"],
    "Liegas#Liego": ["Maka Albarn#0202"],
}
# Ganks : passages dans une lane qui n'est pas la sienne, relevés minute par minute (positions de la timeline)
GANK_DEBUT = 3 # première minute observée (avant : first clear, pas de gank)
GANK_FIN = MINUTE # dernière minute observée (incluse)
GANK_MARGE = 60 # secondes avant/après un passage pendant lesquelles un kill y est rattaché
# Invades : actions (kill/assist, ward, camp) dans le territoire ennemi = derrière la ligne de ses tours encore vivantes
INVADÉ_DEBUT = 0 # première minute observée
INVADÉ_FIN = MINUTE # dernière minute observée (incluse)
INVADÉ_MARGE = 700 # unités : ligne des tours poussée vers la rivière (sinon les raptors, plus avancées que la tour mid, seraient chez l'adversaire)
INVADÉ_ECARTE = 60 # secondes max entre deux actions d'un même invade
# Stats avancées
DIVE_PORTE = 900 # unités : un kill à moins de cette distance d'une tour debout = dive
SEUIL_THROW = 4000 # gold d'avance (ou de retard) d'équipe atteint à un moment de la game pour parler de throw (ou comeback)
FORME_GAMES = 5 # dernières games prises en compte pour la forme (onglets Global)
```

3.2 Étape 2 — Récupérer les games et alimenter le cache

Détail technique

main() charge d'abord les données Data Dragon (noms français, objets, sorts, runes, version du patch), récupère le PUUID de chaque joueur et de ses comptes secondaires, puis tous les ids de games depuis DATE_DEBUT dans les FILES choisies.

- api_get gère les retries, l'erreur 429 (pause selon l'en-tête Retry-After) et la clé expirée.
- Chaque game (match + timeline) est écrite dans cache/<id>.json.gz, de façon atomique, et n'est plus jamais téléchargée.
- Le premier lancement est long à cause des limites de l'API ; les suivants sont rapides grâce au cache.

```
$ python main.py
```

```
PS C:\laragon\www\RAID> python main.py
33 games jouées à 5 ensemble (onglets Ronin)

Fichier généré : stats_RONIN.xlsx
Données du site générées : data/site.json
Pense à remplir les WR en jaune dans l'onglet Matchups (Émeraude+, patch actuel).
```

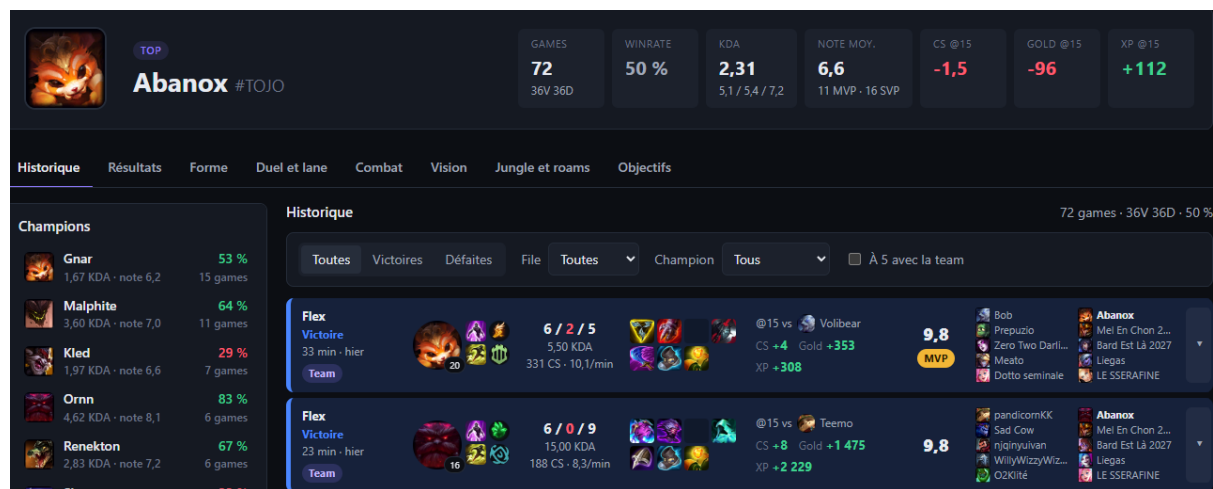
3.3 Étape 3 — Extraire la phase de lane à 15 minutes

Détail technique

extraire_game assemble toutes les stats d'une game pour un joueur, ou renvoie None si la game est écartée : remake, FF avant 15 min, adversaire de lane introuvable, ou poste différent de celui fixé dans POSTES.

Règle	Définition
Lane gagnée	Devant sur au moins 3 des 5 critères à 15 min : CS, gold, XP, vision early, KDA ratio
Vision early gagnée	Devant sur au moins 2 des 3 : wards posées, détruites, pinks
KDA ratio	(Kills + Assists) / max(Deaths, 1)
First clear	Temps pour finir les 6 premiers camps sans back (24 CS jungle), affiné par le passage niveau 4
Duel favorable	WR du matchup > 50 % (WR saisi à la main dans l'Excel)

Note — La vision early sert de critère de vision parce que le vision score n'existe qu'en fin de game dans l'API Riot.



3.4 Étape 4 — Détecter les ganks et les invades

Détail technique

Les ganks et les invades sont calculés pour les 5 joueurs et pour leur adversaire de lane, sans aucun appel API supplémentaire : tout vient de la timeline en cache (positions minute par minute et événements CHAMPION_KILL).

- Zones de lane : polygones ZONES_LANES (TOP, MID, BOT), embouchure de rivière incluse, jungle et objectifs exclus.
- Gank : présence dans une lane étrangère entre GANK_DEBUT et GANK_FIN ; réussi si l'équipe fait plus de kills qu'elle n'a de morts dans la lane pendant la fenêtre.
- Rattrapage : un kill en lane étrangère non rattaché à un passage relevé compte comme un gank de plus.
- Invade : action en territoire ennemi (kill/assist, ward, camp volé), le territoire étant délimité par la ligne des tours encore vivantes, poussée de INVADE_MARGE vers la rivière.

Ces détections se vérifient sur une game précise grâce aux commandes de debug, qui lisent uniquement le cache (pas besoin de clé API) :

```
$ python main.py --zones EUW1_1234567890
```

```
$ python main.py --ganks EUW1_1234567890 "Pseudo#TAG"
$ python main.py --invades EUW1_1234567890
```

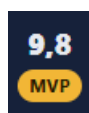
3.5 Étape 5 — Noter la game et désigner MVP / SVP

Détail technique

Chaque game donne une note sur 10 aux 10 joueurs. Le calcul reconstitue le score de DPM.lol, qui n'est pas public, à partir de 149 scores relevés sur 15 de nos games.

- points_dpm : points par onglet (Global, vs Opponent, Objectives, Team, onglet du poste) avec les coefficients relevés sur DPM.
- score_dpm : $1,108 \times \text{total des points} + \text{constante du poste (DPM_PENTE, DPM_CONSTANTE, ajustées par régression)}$.
- note_dpm : score plafonné entre 0 et 100, divisé par 10.
- statuts_mvp : MVP = meilleure note de l'équipe gagnante, SVP = meilleure note de l'équipe perdante ; à note égale, tous les joueurs concernés le sont.

Note — Précision mesurée : 4 points d'écart en moyenne (sur 100) avec le score DPM, et le même meilleur joueur par équipe dans 28 cas sur 30.



3.6 Étape 6 — Générer l'Excel et saisir les WR de matchups

Détail technique

construire_excel écrit stats_ROMIN.xlsx. Toutes les stats sont des formules Excel, qui se recalculent dès qu'un WR est modifié.

Onglet	Contenu
Matchups	Chaque duel rencontré, lien DPM, WR à saisir (colonne jaune), notre WR et l'écart
Games <pseudo>	Une ligne par game, colonnes groupées par thème
Champions <pseudo>	Stats par champion (WR, note, MVP / SVP, moyennes des écarts)
Global <pseudo>	Synthèse par thème : Résultats, Forme, Duel et lane, Combat, Vision, Jungle et roams, Objectifs
Games / Champions / Global Ronin	Les mêmes onglets, uniquement pour les games jouées à 5

Action manuelle : ouvrir le lien DPM de chaque duel et saisir le WR (Émeraude+, patch actuel) dans la colonne jaune de l'onglet Matchups. La saisie est conservée aux lancements suivants et se répercute dans tous les onglets.

Champion	Adversaire	WR duel (DPM)	Nb games	Victoires	Notre WR	Écart vs DPM	Lien
Jinx	Kai'Sa	53,9%	7	6	85,7%	+31,8%	Ouvrir DPM
Jinx	Yunara	51,8%	7	4	57,1%	+5,3%	Ouvrir DPM
Jinx	Draven	52,6%	5	4	80,0%	+27,4%	Ouvrir DPM
Jarvan IV	Ekko	49,3%	3	0	0,0%	-49,3%	Ouvrir DPM
Jinx	Ezreal	56,0%	3	2	66,7%	+10,7%	Ouvrir DPM
Jinx	Jhin	52,8%	3	2	66,7%	+13,9%	Ouvrir DPM
Jinx	Smolder	54,2%	3	2	66,7%	+12,5%	Ouvrir DPM
Jinx	Yasuo	51,3%	3	2	66,7%	+15,4%	Ouvrir DPM

3.7 Étape 7 — Exporter les données et servir le site

Détail technique

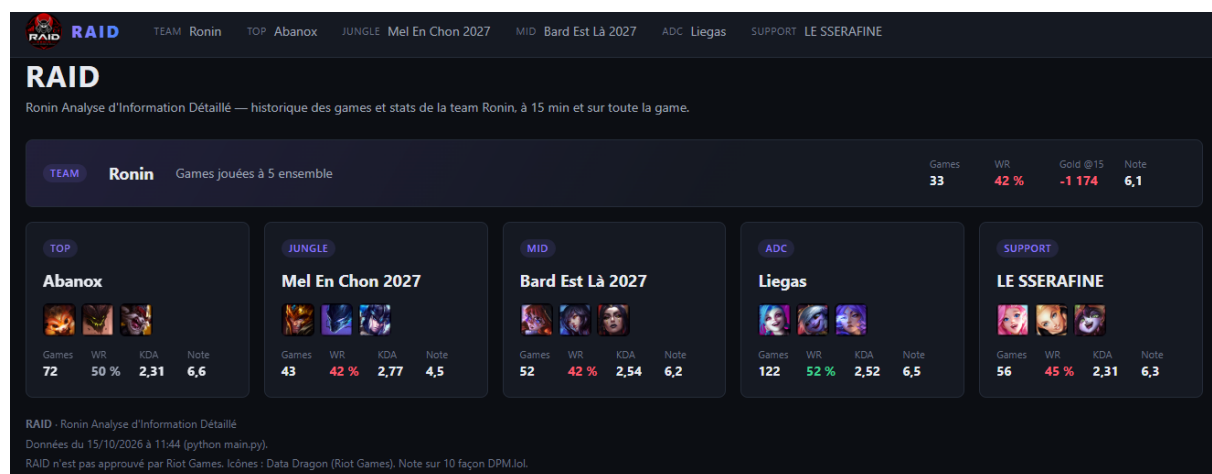
exporter_web écrit data/site.json, qui contient tout ce que le site affiche, déjà calculé. Le site relit ce fichier à chaque requête : relancer python main.py suffit à le mettre à jour.

Dans Laragon : Menu > Apache > Reload. Le virtual host `http://raid.test` pointe sur `public/`. Sans Laragon, le serveur intégré de PHP suffit :

```
$ php -S localhost:8000 -t public # http://localhost:8000
```

URI	Page	Rôle
/	accueil	L'équipe et les 5 joueurs
/equipe	equipe	Games à 5 et onglets d'équipe (404 si aucune)
/equipe/game/<id>	détail_equipe	Fragment des 10 joueurs, chargé par le JS
/joueur/<slug>	joueur	Historique filtrable, champions, onglets par thème
/joueur/<slug>/game/<id>	détail	Fragment des 10 joueurs, joueur surligné
tout le reste	introuvable	404

Note — Si data/site.json est absent ou illisible, toutes les URL répondent 503 avec une page invitant à lancer python main.py.



3.8 Étape 8 — Vérifier avec les tests

Détail technique

Trois suites, une par langage, sans aucun appel réseau (API Riot et Data Dragon mockés) ni accès au vrai cache, aux .xlsx ou à data/site.json.

```
$ python -m pytest # 353 tests : calculs, Excel, export
$ vendor/bin/phpunit # 108 tests : pages, routes, format
$ npm test # 50 tests : filtres, onglets, graphes
```

extraire_game est testé sur 4 vrais matchs (tests/fixtures/) dont la sortie est comparée à un snapshot. Après un changement volontaire d'un calcul, vérifier le diff puis régénérer :

```
$ MAJ_SNAPSHOTS=1 python -m pytest tests/test_extraire_game.py
```

Astuce — Pour faire évoluer un calcul : modifier main.py (section concernée), relancer les tests et régénérer les snapshots si le changement est voulu, puis relancer python main.py. Le PHP ne contient aucun calcul et n'a pas à être touché.

4. Structure / Données

Pas de base de données ni de comptes utilisateurs : trois fichiers générés, le cache étant la source de vérité.

Élément	Description	Remarque
cache/<id>.json.gz	Match + timeline bruts de l'API Riot, un fichier par game	Ignoré par Git ; supprimer le dossier pour repartir de zéro
stats_RONIN.xlsx	Onglets en formules + WR saisis à la main	Supprimer le fichier réinitialise les WR saisis
data/site.json	Tout ce que le site affiche, déjà calculé	Ignoré par Git ; relu à chaque requête

4.1 Clés de data/site.json

Clé	Type	Contenu
genere	date ISO	Date de génération, affichée en pied de page
equipe / minute / forme_games	texte / entier	NOM_EQUIPE, MINUTE (15), FORME_GAMES (5)
ddragon, sorts, runes	texte / objets	Version et images Data Dragon
joueurs	liste	slug, riot_id, poste, resume, champions, games, themes
matches	objet	id de game → les 10 joueurs (champion, K/D/A, objets, note, MVP...)
equipe_stats	objet / null	resume, courbe_gold, games à 5, themes

Un thème (resultats, forme, lane, combat, vision, jungle, objectifs) est une liste de tableaux décrits par leur titre, leurs colonnes (libellé, format, couleur) et leurs lignes. La fixture tests/php/fixtures/site.json a la même forme ; un test pytest vérifie qu'elle a exactement les clés exportées.

4.2 Arborescence utile

Chemin	Rôle
main.py	Tout le calcul : API, cache, stats, Excel, export du site
solo_abanox.py, solo_adam.py, solo_sinrha.py	Variante SoloQ d'un joueur (stats_solo_*.xlsx, pas de site)
src/	Back-end du site (App, Routeur, Site, Tableau, Format, Icones, Vue)
templates/	Gabarit et pages PHP natives, échappement par e()
public/	Racine web : index.php, .htaccess, assets CSS / JS / images
tests/	pytest, tests/php (PHPUnit), tests/js (Vitest)

5. Déploiement / CI-CD

RAID n'a ni pipeline CI/CD ni hébergement distant : c'est un outil interne exécuté en local. Le « déploiement » se résume à régénérer les données puis recharger le site.

- Régénérer la clé de dev Riot si elle a plus de 24 h et la mettre dans `.env`.
- Lancer `python main.py` : nouvelles games téléchargées, Excel et `data/site.json` régénérés.
- Recharger Apache dans Laragon (première fois) ; ensuite le site relit le JSON tout seul.
- Saisir dans Matchups les WR des nouveaux duels, puis relancer `main.py` pour qu'ils soient pris en compte.

Note — Mise en ligne sur un serveur et pipeline de tests automatique : [NON VÉRIFIABLE : aucun workflow ni configuration serveur dans le dépôt]. Un pipeline GitHub Actions lançant les trois suites de tests serait l'évolution naturelle.

6. Points de vigilance

Attention — La clé API de développement Riot expire au bout de 24 h : la régénérer avant chaque lancement, sinon `api_get` s'arrête sur une erreur de clé.

Attention — Clé API uniquement dans `.env` (ignoré par Git) : ne jamais la commiter. Le site ne l'utilise jamais et n'appelle pas l'API.

Attention — Les positions ne sont relevées qu'une fois par minute : les ganks ratés sont sous-estimés, et les wards et camps volés des invades sont des positions estimées.

Attention — La note sur 10 est une reconstitution du score DPM.lol : pour l'affiner, relever les scores DPM de nouvelles games et réajuster `DPM_PENTE` et `DPM_CONSTANTE`.

Attention — Les scripts `solo_*.py` n'écrivent pas `data/site.json` (`FICHER_WEB = None`) : le site garde les données de l'équipe.

Attention — Le `.htaccess` racine fait `Require all denied` : seul `public/` est servi, ce qui protège `.env`, `cache/` et `data/` même si le virtual host pointe sur la racine du projet.

Attention — `Chart.js` est chargé depuis `jsDelivr` sans attribut `integrity`, et aucune CSP n'est définie.

Attention — Le WR de matchup (duel) n'existe que dans l'Excel : il n'est pas encore affiché sur le site.