AETHER

Action Plan EIP

High-performance web browser in Rust

Contents

Part 1 — Contexte 2
1.1 Origin of the project 2
1.2 Problem identified 2
1.3 The solution: Aether 2
1.4 Project Objectives 2
1.5 Inside Track selected 3
Part 2 — Technical Specifications 3
2.1 Technology Stack 3
2.2 Project Architecture 3
2.3 User Stories 4
Navigation de base 4
Sécurité et vie privée 4
Performance et paramétrage 4
2.4 Milestones 5
Part 3 — Non-Technical Specifications 6
3.1 Objectif obligatoire — Évaluation et intégration de nouvelles technologies (Veille technologique) 6
Ce que nous allons faire 6
Objectifs mesurables 7
3.2 Objectif complémentaire #1 — Protéger et valoriser la technologie 7
Protection légale 7
Protection technique 7
Valorisation externe 8
3.3 Objectif complémentaire #4 — Mesurer, tester et optimiser les performances techniques 8
Indicateurs de performance retenus 8
Protocole de tests et d'optimisation 8
Processus d'optimisation 9
3.4 Synthèse des engagements 9

Part 1 — Contexte

1.1 Origin of the project

Aether was born from the observation that modern web browsers have become extremely resource-intensive and often opaque regarding their privacy practices. Chrome, Firefox, and their derivatives contain millions of lines of C and C++ code, dozens of background processes, and integrate data collection mechanisms by default for the benefit of their respective developers.

Faced with this situation, we decided to design a Servo-based web browser in Rust, a modern system language that offers memory security guarantees without relying on a garbage collector. The goal is to offer a lighter, faster, and more user-friendly alternative.

1.2 Problem identified

The problem
Current browsers suffer from three major problems: (1) a high memory and CPU footprint, degrading performance on modest machines, (2) a lack of transparency on user data collection, and (3) the absence of effective built-in protection against ads and trackers, forcing users to install third-party extensions.

1.3 The solution: Aether

Aether is an open-source desktop browser developed in Rust. It uses the Servo rendering engine (Linux Foundation) for displaying web pages and egui for its native graphical interface. It natively integrates an ad and tracker blocking engine, enabling faster and more private browsing without additional configuration.

- Primary language: Rust (memory safe, native performance, no garbage collection)
- Rendering engine: Servo (experimental web engine in Rust developed by Mozilla and adopted by Linux)
- Graphical interface: egui (out-of-the-box GUI library in Rust)
- Integrated blocking: ad and tracker filtering engine (compatible with EasyList lists)
- Target platforms: Windows, macOS, and Linux (cross-platform desktop)

1.4 Project Objectives

The primary objective is to deliver a functional, high-performance, and secure web browser capable of navigating common websites with measurably superior performance compared to mainstream browsers on modest hardware configurations.

Secondary objectives include implementing a modular architecture that allows for external contributions, comprehensively documenting the technical choices, and demonstrating concrete performance gains through rigorous benchmarking.

1.5 Inside Track selected

Technical Track
The Aether project is part of the Technical Track. The additional objectives selected are: #1 Protect and promote the technology (open-source license, protection repository, dissemination strategy) and #4 Measure, test and optimize technical performance (memory/CPU/rendering time benchmarks, load tests, targeted optimizations).

Part 2 — Technical Specifications

2.1 Technology Stack

The choice of technology stack was guided by the objectives of performance, security and long-term maintainability.

Composant	Technologie	Justification
Langage	Rust (stable)	Sécurité mémoire compile-time, zero-cost abstractions, pas de GC
Moteur de rendu	Servo	Moteur HTML/CSS en Rust, évite la dépendance à Chromium/Gecko
Interface graphique	egui	GUI immédiate en Rust pur, légère et cross-platform
Build system	Cargo	Gestionnaire de paquets et build tool natif Rust
CI/CD	GitHub Actions	Intégration continue multi-plateforme (Linux, macOS, Windows)
Tests	cargo test + criterion	Tests unitaires natifs + benchmarks de performance Rust
Linting/Qualité	Clippy + rustfmt	Analyse statique et formatage de code standard Rust
Blocage pubs	Moteur custom (filtres EasyList)	Contrôle total, performance maximale, pas de dépendance externe

2.2 Project Architecture

Aether suit une architecture modulaire en crates Rust indépendantes, permettant de tester, maintenir et faire évoluer chaque composant séparément. Voici les modules principaux :

- aether-core : orchestrateur principal, gestion des onglets, navigation, historique
- aether-renderer : intégration Servo pour le rendu HTML/CSS/JS
- aether-ui : interface graphique egui (barre d'adresse, onglets, panneau de contrôle)
- aether-net : couche réseau (requêtes HTTP/HTTPS, gestion des certificats TLS)

- aether-adblock : moteur de filtrage des publicités et traceurs
- aether-config : gestion de la configuration utilisateur persistante
- aether-bench : suite de benchmarks et tests de performance

2.3 User Stories

Navigation de base

ID	En tant que...	Je veux...	Afin de...	Priorité
US-01	utilisateur	saisir une URL et charger une page web	je puisse naviguer sur internet	Haute
US-02	utilisateur	ouvrir plusieurs onglets simultanément	je puisse consulter plusieurs pages en parallèle	Haute
US-03	utilisateur	utiliser les boutons précédent/suivant/recharger	je puisse naviguer dans l'historique de navigation	Haute
US-04	utilisateur	effectuer une recherche depuis la barre d'adresse	je n'aie pas besoin d'aller sur un moteur de recherche séparément	Haute
US-05	utilisateur	consulter mon historique de navigation	je puisse retrouver des pages visitées précédemment	Moyenne
US-06	utilisateur	ajouter des pages en favoris	je puisse y accéder rapidement	Moyenne

Sécurité et vie privée

ID	En tant que...	Je veux...	Afin de...	Priorité
US-07	utilisateur	bénéficier d'un blocage automatique des publicités	mes pages chargent plus vite sans publicités intrusives	Haute
US-08	utilisateur	bénéficier d'un blocage des traceurs tiers	mes données de navigation ne soient pas collectées	Haute
US-09	utilisateur	voir un indicateur de connexion sécurisée (HTTPS)	je sache si ma connexion est chiffrée	Haute
US-10	utilisateur	voir un récapitulatif des éléments bloqués par page	je sache combien de traceurs ont été bloqués	Moyenne
US-11	utilisateur	désactiver le bloqueur pour un site spécifique	je puisse accéder à des contenus qui nécessitent les pubs	Moyenne

Performance et paramétrage

ID	En tant que...	Je veux...	Afin de...	Priorité
US-12	utilisateur	voir la consommation mémoire et CPU du navigateur	je puisse vérifier son efficacité en temps réel	Moyenne
US-13	utilisateur	configurer le moteur de recherche par défaut	j'utilise le moteur que je préfère	Basse
US-14	utilisateur	importer/exporter mes favoris	je puisse migrer depuis un autre navigateur	Basse
US-15	développeur	accéder à des logs de performance détaillés	je puisse identifier et corriger les goulots d'étranglement	Haute

2.4 Milestones

Milestone	Période	Objectifs	Critères de validation
M1	Sept. – Oct. 2025	• POC : rendu d'une page HTML simple via Servo • UI de base avec egui (barre d'adresse, onglet unique) • Chargement HTTP/HTTPS fonctionnel • README initial + architecture documentée	• Affichage de wikipedia.org sans crash • Compilation stable sur Linux • Diagramme d'architecture v1 livré
M2	Nov. – Déc. 2025	• Gestion multi-onglets • Navigation historique (précédent/suivant) • Intégration du moteur adblock (listes EasyList) • Premiers benchmarks mémoire/CPU vs Firefox	• 10 onglets ouverts simultanément sans crash • Blocage confirmé sur 95%+ des pubs EasyList • Rapport de benchmark v1 disponible
M3	Janv. – Févr. 2026	• Cross-compilation Windows et macOS • Suite de tests unitaires sur les modules critiques • Optimisation mémoire : réduction de 20% vs M2 • Dashboard de métriques intégré (perf panel)	• Build fonctionnel sur les 3 OS cibles • Couverture de tests > 60% sur aether-core • Gain de 20% RAM mesuré et documenté

Milestone	Période	Objectifs	Critères de validation
M4	Mars – Avr. 2026	• Dépôt légal du code (enveloppe Soleau numérique) • Documentation technique complète (wiki GitHub) • Audit de sécurité interne (Cargo audit, analyse statique) • Publication d'un article de benchmark public	• Preuve de dépôt reçue • Wiki avec 5+ pages techniques détaillées • 0 vulnérabilité critique détectée par cargo audit
M5	Mai – Juin 2026	• Version beta publique sur GitHub • Intégration d'au moins une contribution externe • Benchmarks finaux complets (latence, RAM, CPU, rendu) • Démonstration publique / présentation EIP	• Release GitHub avec binaires téléchargeables • Au moins 1 PR externe acceptée • Rapport de performance final livré

Part 3 — Non-Technical Specifications

Cette partie présente les intentions sur les objectifs non-techniques du Technical Track, conformément aux exigences du programme EIP. Les objectifs obligatoires et les deux objectifs complémentaires retenus (#1 et #4) sont détaillés ci-dessous.

3.1 Objectif obligatoire — Évaluation et intégration de nouvelles technologies (Veille technologique)

Note d'intention
La veille technologique est centrale dans le développement d'Aether. Le domaine des navigateurs web et du langage Rust évolue rapidement, avec des acteurs comme le projet Servo, l'écosystème WebAssembly, et les nouveaux standards W3C. Il est crucial de rester à la pointe pour prendre des décisions architecturales éclairées.

Ce que nous allons faire

- Mettre en place une veille hebdomadaire sur les sources suivantes :
- This Week in Rust (newsletter hebdomadaire)
- Blog officiel Servo (servo.org/blog)
- GitHub : suivi des releases de Servo, egui, tokio

- Reddit r/rust, r/programming, forums internals.rust-lang.org
- Arxiv/Semantic Scholar pour les publications sur les moteurs de rendu
- Rédiger des notes comparatives sur les alternatives évaluées (ex : Tauri vs egui, WebKit vs Servo)
- Réaliser 1 à 2 POC d'intégration de nouvelles dépendances avant tout merge en main
- Documenter toutes les décisions techniques dans un ADR (Architecture Decision Record) sur le wiki GitHub
- Partager nos avancées sur des forums spécialisés (This Week in Rust, r/rust) pour obtenir des retours

Objectifs mesurables

- Minimum 1 note de veille rédigée par mois
- Au moins 2 benchmarks comparatifs réalisés sur l'année (ex : Servo vs WebView2, egui vs Tauri)
- Au moins 1 nouvelle technologie intégrée avec justification documentée
- Au moins 2 contributions à des discussions communautaires (GitHub issues, forums)

3.2 Objectif complémentaire #1 — Protéger et valoriser la technologie

---

Note d'intention
Aether représente un investissement technique significatif en termes d'architecture, d'implémentation du moteur adblock en Rust et d'intégration de Servo. Il est important de protéger ce travail tout en le valorisant auprès de la communauté open-source et de recruteurs potentiels.

Protection légale

- Dépôt du code source via l'Agence pour la Protection des Programmes (APP) ou équivalent numérique (dépôt horodaté sur un tiers de confiance)
- Rédaction et application d'une licence open-source : choix entre MIT (permissive, facilite les contributions) et Apache 2.0 (inclut protection contre les brevets) — décision documentée dans un ADR
- Mise en place d'un fichier CONTRIBUTORS.md et d'un accord de contribution (CLA light) pour les contributions externes

Protection technique

- Audit de sécurité avec cargo audit (vulnérabilités dans les dépendances) à chaque release
- Analyse statique avec Clippy en mode strict (deny warnings) intégrée à la CI
- Chiffrement des données de configuration utilisateur sensibles (certificats, cookies)

- Documentation d'une threat model simplifiée : identification des surfaces d'attaque (réseau, fichiers de config, extensions futures)

Valorisation externe

- Rédaction d'une page technique (tech brief) présentant l'architecture d'Aether, ses innovations et ses performances, destinée à des recruteurs et partenaires potentiels
- Publication de démonstrations de performance sur GitHub (benchmarks, GIFs de l'UI, captures de profiling)
- Participation à un événement ou meetup Rust (ex : RustFest, meetup Rust Paris) pour présenter le projet

3.3 Objectif complémentaire #4 — Mesurer, tester et optimiser les performances techniques

---

Note d'intention
La performance est la promesse centrale d'Aether. Il ne suffit pas d'affirmer qu'Aether est plus rapide ou plus léger : chaque amélioration doit être prouvée par des mesures objectives, reproductibles et comparées à des références comme Firefox ou Chromium.

Indicateurs de performance retenus

Indicateur	Description	Outil de mesure
Consommation RAM	Mémoire RSS à chargement d'une page type (Wikipedia, GitHub)	heaptrack, /proc/meminfo, valgrind
Temps de rendu (First Contentful Paint)	Temps entre la requête HTTP et l'affichage du premier contenu	Instrumentation interne + logs Servo
Utilisation CPU	% CPU moyen lors du scrolling et du chargement de page	perf, cargo flamegraph
Efficacité du bloqueur	Nombre de requêtes bloquées / temps de traitement par requête	Tests sur corpus EasyList (10 000 règles)
Taille binaire	Taille de l'exécutable final par plateforme	cargo bloat, size tools

Protocole de tests et d'optimisation

- Tests de performance automatisés intégrés à la CI (cargo bench avec criterion) — exécutés à chaque PR
- Benchmarks comparatifs mensuels versus Firefox en navigation standard (corpus de 10 pages représentatives)
- Tests de stress : chargement de 20 onglets simultanément, mesure de la stabilité et de la dégradation des performances

- Tests de résilience : simulation de pannes réseau, pages malformées, certificats expirés
- Analyse des flamegraphs à chaque milestone pour identifier les hot spots

Processus d'optimisation

- Chaque optimisation sera précédée d'une mesure baseline et suivie d'une mesure post-optimisation
- Les trade-offs (performance vs lisibilité du code, performance vs sécurité) seront documentés dans les ADR
- Un rapport de performance sera publié à chaque milestone (M2 à M5) avec graphiques avant/après
- Objectif chiffré : empreinte mémoire inférieure de 30% à Firefox sur un scénario de navigation standard avec 5 onglets ouverts

3.4 Synthèse des engagements

Objectif	Livrable clé	Échéance cible
Veille tech (obligatoire)	Notes de veille mensuelles + 2 benchmarks comparatifs + 1 intégration documentée	Tout au long de l'année
Architecture documentée (obligatoire)	README complet + diagrammes + ADR + wiki GitHub	M1 → M3
Protection & valorisation (#1)	Dépôt légal + licence + audit sécurité + tech brief	M4
Mesure & optimisation (#4)	Suite de benchmarks CI + rapports avant/après + flamegraphs	M2 → M5

Document Date: September 15, 2026

Modify Date: November 13, 2026