MadEasy AI Browser – Komplett Plan og Arkitektur (Windows & Android) (MVP → V1)

Plattform: Windows PC-program og Android-app (standalone, ikke plugin).

Elevator pitch: En AI-drevet nettleser bygget fra bunnen av hvor brukeren beskriver målet i et chatfelt – MadEasy AI Browser planlegger, navigerer, klikker, skriver, laster ned og rapporterer ferdig resultat. Den kan utføre komplekse workflows, samle leads, publisere innhold, og fungere som prosjektleder i utviklingsløp sammen med dev-AI-er som Lovable, Bolt og Replit.

---

1) Mål og kjerneevner

- Automatisere nettoppgaver: skjema, scraping, publisering, QA.
- Forstå formål og definere delmål/akseptkriterier.
- Dokumentere med skjermbilder, logger, rapporter.
- Samhandle med dev-AI og lede prosjekter.

Use cases: Lead-innsamling, skjemautfylling, research, innholdspublisering, nedlasting/sortering, QA-testing.

---

2) Brukeropplevelse (UX)

- **Chat-first kontroll:** Beskriv målet → MadEasy AI lager plan → bruker godkjenner.
- **Live Steps-panel:** viser handlinger og skjermbilder.
- **Tillatelser (Scopes):** lese, skrive, nedlast, opplast, login – per domene/økt.
- **Rapporter:** HTML/PDF med logg, kilder, vedlegg.
- **Playbook-eksport:** YAML/JSON for senere gjenbruk.

---

3) Arkitektur

- **Shell:** C# .NET 8 (Windows), Kotlin/Java (Android).
- **Render-motor:** CEF (Windows) og WebView (Android).
- **Agent-lag:** Planner → Critic → Executor → PM Agent.
- **Tools:** BrowserTool (CDP), VisionTool (OCR/matching), FormFiller, FileTool, CodeTool, DevBridge, CRM/LeadTool.
- **Memory:** SQLite + embeddings, Secrets Vault.
- **Integrasjoner:** Gmail/M365, lokale LLM-er, API-er.

Arkitekturdiagram

---

4) Workflow-DLS (YAML)

Eksempel:

```yaml
name: leads_cast_iron_eu
inputs:
  query: "cast iron cookware wholesaler EU"
  take: 50
steps:
  - goto: https://www.google.com
  - search: ${query}
  - scrape_results: { take: ${take}, fields: [company, url, email, phone] }
  - export: { type: xlsx, path: /exports/leads.xlsx }
```

---

5) Lead-motor

- Kilder: søkemotorer, kataloger, schema.org.
- Metoder: regex + semantikk, MX/SMTP validering.
- Scoring: fit × reach × intent × credibility.
- Output: XLSX/CSV + rapport.

---

6) DevBridge & prosjektledelse

Mål: La **MadEasy AI Browser** følge og **lede utviklingen** av et prosjekt **gjennom chat-grensesnittene** på Lovable, Bolt, Replit og mgx.dev – akkurat slik en menneskelig utvikler/PM ville gjort. Ingen proprietære «hemmelige API-er» er påkrevd i MVP; alt kan kjøres via nettleser-UI med trygg automasjon.

6.1 Hvordan det virker (chat-først orkestrering)

1) **Work Order (WO)** definerer mål, akseptkriterier og risiko. 2) **Context Pack (CP)** gir repo/filer/lenker/secrets. 3) **Chat-Adapter** åpner plattformen i en tab, finner chat/kommandofeltet, og sender strukturerte meldinger basert på WO+CP. 4) **Observér & forstå**: MadEasy AI leser svar fra dev-AI (kodeforslag, logger, feilmeldinger), oppdaterer status og foreslår neste steg. 5) **Handle**: ber dev-AI om endringer, ber om bygg/test, eller klikker UI-knapper (Run/Preview/PR/Deploy) når nødvendig. 6) **Valider**: kjør asserts (build ok, tests ok, ruter finnes, Lighthouse ≥ terskel). Loop til aksept er oppfylt eller eskaler.

6.2 Menneskelik interaksjon via UI

- **Skriv/les i chat** med naturlig språk, men med *operativ struktur*: mål → underoppgaver → sjekkpunkter.
- **Høflige, tydelige prompts** med bulletpunkter og akseptkriterier.
- **Skjermlesing**: AI skanner kode-/loggpanel, oppsummerer og siterer relevante biter før den beslutter neste prompt.

- **Klikk & tast** i UI: Run/Preview/Commit/PR/Deploy håndteres med sikre selectorer (ARIA→tekst→CSS fallback).

6.3 Adaptere (Lovable, Bolt, Replit, mgx.dev)

Felles Chat-Adapter API (UI-automatisering, ikke API-bindinger nødvendig i MVP): - `open_workspace(project|template)` - `focus_chat()` / `send_message(text)` / `await_response(timeout)` - `open_file(path)` / `apply_patch(diff)` - `run_build()` / `open_preview()` / `extract_preview_url()` - `open_pr()` / `deploy()` - `read_logs(kind)`

Selektor-policy: ARIA > synlig tekst > stabile data-attributes > robust CSS. Fallback-læring lagres per domene.

6.4 PM-intelligens i loopen

- **Plan Builder** bryter WO til etapper med **Quality Gates** (build, tests, perf, DoD).
- **Decision Log (ADR)**: hver større beslutning logges med alternativer og begrunnelse.
- **Risk Radar**: oppdateres automatisk (f.eks. fallende LH-score, testsvikt, store diffs).
- **Standups**: generer «I går / I dag / Hindringer» fra faktisk interaksjon i chat og CI.

6.5 Sikkerhet & kontroll

- **Human-in-the-loop** for høyrisiko (publisering, secrets, store migrasjoner).
- **Secrets Vault**: injiser kun ved behov (release-once per økt), maskér i logger/skjermbilder.
- **Tillatelser per domene**: eksplisitte «skriv/last opp/commit/deploy»-scopes.

6.6 Validering (akseptkriterier → asserts)

Eksempler: `build:success`, `tests:pass`, `route:/contact exists`, `lighthouse.perf>=85`. Feil ⇒ auto-retry m/ forbedringsprompt, ellers eskaler med forslag.

6.7 Eksempel: Chat-drevet flyt (plattform-agnostisk)

Prompt-mal (sendes i chat-UI):

> Goal: Build Next.js landing with CTA + /contact (email validation). Acceptance: build ok, tests ok, route /contact, Lighthouse ≥ 85 desktop. Steps now: 1) create baseline (Next+Tailwind) 2) implement CTA and /contact 3) run build & preview 4) share preview URL. Constraints: minimal deps, accessible UI (WCAG AA).

Oppfølging (automatisk): - «Run build, then share preview URL.» - «If build fails: show error summary + propose fix; apply and retry.» - «Optimize for performance to reach Lighthouse ≥ 85; lazy-load heavy components.»

6.8 YAML-playbook (chat-orkestrering)

```yaml
name: dev_ai_chat_orchestration
adapter: chat
inputs:
  platform: bolt # lovable|replit|mgx
```

```
  brand: "MadEasy"
  perf_min: 85
steps:
  - goto: https://${platform}.new
  - focus_chat: {}
  - send_message:
      text: |
        Goal: Build a Next.js landing for ${brand} with Tailwind.
        Acceptance: build ok, tests ok, route /contact, Lighthouse ≥ $
{perf_min} (desktop).
        Steps: 1) baseline 2) CTA + /contact 3) run build & share preview.
        Constraints: minimal deps, accessible (WCAG AA).
  - await_response: { timeout: 600000 }
  - click: { selector: 'aria/Run' }
  - extract_preview_url: { save_as: preview_url }
  - validate:
      asserts: [ 'build:success', 'route:/contact exists' ]
  - measure_lighthouse: { url: ${preview_url}, min_perf: ${perf_min} }
  - if: ${metrics.lighthouse.perf} < ${perf_min}
    then:
      - focus_chat: {}
      - send_message:
          text: |
            Performance is below ${perf_min}. Please optimize bundle size,
lazy-load non-critical components, and ensure images use next/image.
      - click: { selector: 'aria/Run' }
      - measure_lighthouse: { url: ${preview_url}, min_perf: ${perf_min} }
  - open_pr: {}
  - deploy: {}
  - report: { include: [summary, metrics, pr_url, deploy_url] }
```

6.9 mgx.dev – særtrekk i adapteret

- **Prototyping-modus**: rask generering av funksjoner med tydelig «scope box».
- **Refaktorering**: be mgx om å «explain diff» og «justify change» før commit.
- **Testdrevet**: «generate tests for …», kjør og reparer feil i loop.
- **PR-kvalitet**: mgx skriver PR-tekst med lenker til WO og akseptkriterier.

6.10 Preview-testing & tilbakemeldinger

- Når Lovable, Bolt, Replit eller mgx.dev gir **preview-URL**, åpner MadEasy AI den i egen tab.
- **Validering:** kjør automatiske asserts (ruter finnes, skjema fungerer, Lighthouse/performance ≥ terskel, a11y med Axe-core).
- **Visuell verifisering:** ta skjermbilder og sammenlign med design/akseptkriterier.
- **Funksjonelle tester:** klikk gjennom CTA, fyll skjema, sjekk respons.
- **Tilbakemelding i chat:** AI genererer kort rapport (OK / feil / forslag) og poster tilbake i dev-plattformens chat.
- **Loop:** iterer med dev-AI til preview møter akseptkriterier.

---

7) Sikkerhet

- Secrets Vault, domene-scopes, sandbox-profiler.
- 2FA/TOTP-støtte.
- Personvernfilter i logger.
- Revisjonsspor med hash-kjeding.

---

8) Observability

- Skjermbilder, DOM-utdrag, nettverkslogg.
- Session Replay.
- Rapport med resultatindikatorer og neste steg.

---

9) UI

- **Venstre:** Chat + Plan.
- **Midten:** Live browser.
- **Høyre:** Logg, tillatelser, skjermbilder.
- **Topp:** Profiler, Playbooks, Autonomi-nivå.
- **Dev/PM:** Repo-tre, diff/PR, testlogg, status.

---

10) MVP

- Shell + BrowserTool basics.
- Planner/Executor.
- Skjermbilder + rapport.
- Scraping + eksport.
- Tillatelser v1.
- Lead-motor v1.

Demo: «Finn 20 EU-forhandlere av støpejern, eksporter Excel, lag oppsummering.»

---

11) Roadmap (V1)

- Selektor-ML + visuell fallback.
- Skjemautfylling m/ validering.
- Playbook-editor.
- 2FA/TOTP + Vault GUI.
- Extensions-støtte.
- Avanserte rapporter.
- Human-in-the-loop sjekkpunkter.

---

12) Risiko & mitigasjon

- Skjøre nettsider → fallback + feilhåndtering.
- Innlogging → manuell bekreftelse.
- Anti-bot → pacing, respektere ToS.
- Juridikk → tillatelser, loggføring.

---

13) Teknologistack

- **Frontend:** WPF/WinUI (Windows), Android Jetpack Compose.
- **Backend:** .NET 8, Kotlin.
- **AI:** OpenAI/Anthropic/local LLM.
- **Data:** SQLite, ClosedXML, QuestPDF.
- **Vision:** OCR med ONNX/Tesseract.

---

14) Nettleser-differensiering

- Mål-modus (Browse, Extract, Automate, Review).
- Command Palette (Ctrl+K / Voice).
- Permission chips + Privacy ledger.
- Data Table View.
- Explain Element overlay.
- Session Replay.
- Ephemeral profiler.

---

15) PM-evner

- Plan Builder (WO → delmål, akseptkriterier).
- RACI & Quality Gates.
- Auto-standups, risk radar.
- ADR/Decision Log.

---

16) Samplanlegging & sparring

- Kickoff-canvas.
- Idea Tournament.
- Impact/Effort-matrise.
- Counterfactual-runde.
- Kano-light.
- Spec→Playbook-generator.
- Eksperimentkort.

---

17) GitHub-oppfølging

- Traceability Graph.
- Quality Gates i CI.
- PR-mal med akseptkriterier.
- Branch-beskyttelse.
- Auto-review (MadEasy Spec Compliance Report).
- Release hygiene.

---

18) Forbedrede browser-muligheter

- Voice & multimodal input.
- Smart context switching.
- Collaborative mode.
- Personalized dashboards.
- Plugin marketplace.
- Offline replay.
- Trust & compliance center.

---

19) Anonym & uoppdagbar surfing

- Ephemeral profiler.
- Fingerprint randomization.
- Nettverksanonymisering (Tor/VPN/I2P).
- Isolated containers.
- Tracker & ad-blocking.
- Cloaked automation.
- Audit toggle.
- Auto-expiry av logger og nedlastinger.

---

20) Transkripsjon, oppsummering, logg & arkiv

- STT-transkripsjon av møter/samtaler.
- Event-logg fra handlinger/browsing.
- Auto-oppsummeringer (kort, detalj, executive).
- Arkiv med søk, FTS5, kategorisering.
- Eksport til PDF/HTML/CSV.

---

21) MadEasy-kommentarbot (mad-easy-comment-bot.yml)

GitHub Action som poster **MadEasy Spec Compliance Report** på PR-er.

---

22) Plattformer og teknologi (Windows & Android)

- **Windows:** C# .NET 8, WPF/WinUI 3, CEF/WebView2
- **Android:** Kotlin, Jetpack Compose, WebView
- **Kjerne:** Delt Planner/Critic/Executor/PM Agent via gRPC
- **Playbooks:** YAML felles
- **Profiler:** kryptert Secrets Vault (DPAPI/Keystore)

- **Utviklingssupport:** mgx.dev for AI-drevet utvikling og automatisering

- **Windows:** C# .NET 8, WPF/WinUI 3, CEF/WebView2

- **Android:** Kotlin, Jetpack Compose, WebView
- **Kjerne:** Delt Planner/Critic/Executor/PM Agent via gRPC
- **Playbooks:** YAML felles
- **Profiler:** kryptert Secrets Vault (DPAPI/Keystore)

Oppsummering: MadEasy AI Browser er en tverrplattform (Windows + Android) nettleser med AI-styring, anonymitet, lead-motor, DevBridge, observability og prosjektleder-funksjoner – bygget for både produktivitet og sikkerhet.

---

23) Arkitekturdiagram

MadEasy AI Browser Architecture

---

24) Preview-drevet testing & tilbakemeldinger

Mål: Når Lovable/Bolt/Replit/mgx.dev eksponerer **Preview/Live URL**, skal MadEasy AI automatisk **forstå, teste og gi tilbakemeldinger** på det som er utviklet – akkurat som en menneskelig QA/PM.

24.1 Oppdagelse av preview

- **Auto-ekstraksjon:** finn `Preview`-knapp/lenke i UI (ARIA/tekst), les `href`.
- **Heuristikk:** valider at URL svarer 200, og identifiser ruter (`/`, `/contact`, osv.) via sitemap/links crawl (dybde 1–2).
- **Kontekst:** bind preview-URL til gjeldende WO/PR for sporing.

24.2 Testbatteri (MVP → V1)

- **Bygg/generelt:** `build:success`, konsollfeil=0, 3rd-party feil varsles.
- **Ytelse:** Lighthouse desktop (perf ≥ terskel), LCP/TBT/CLS diff vs. base.
- **Tilgjengelighet:** axe-core (kritiske WCAG brudd blokkerer).
- **Ruter:** `route_exists` for avtalte paths; statuskode 200.
- **Forms:** skjemaprolog (påkrevd, e-postvalidering, feilmeldinger, submit-flow).
- **Visuell regresjon (V1):** skjermbilder per side (desktop breakpoint), pikseldiff ≥ terskel flagges.
- **SEO/meta (V1):** title/description, canonical, hreflang (hvis relevant).

24.3 Feedback-sløyfe (chat-drevet)

1) Kjør testbatteriet på preview-URL. 2) **Oppsummer funn** kort (pass/fail + nøkkeletall). 3) **Foreslå konkrete forbedringer** (patch/commit-klar forklaring). 4) **Send høflig, strukturert melding i chat-UI** på plattformen: - «Findings», «Why it matters», «Suggested fix», «Acceptance re-test». 5) Be om ny build/preview og **re-test** automatisk.

Meldingsmal (til dev-AI i chat):

> Findings: Lighthouse perf=78 (<85). Contact form accepts invalid email.
>
> Why it matters: perf impacts conversion; invalid emails pollute CRM.
>
> Suggested fix: lazy-load heavy components; enforce HTML5 + regex email validation.
>
> Please apply the fixes, run build, and share a new preview URL. Acceptance: perf ≥ 85, form rejects `test@invalid`.

24.4 YAML-utdrag (preview-QA)

```
- extract_preview_url: { save_as: preview }
- qa_suite:
    url: ${preview}
    asserts:
      - build:success
      - route_exists: { path: "/contact" }
      - lighthouse: { desktop_perf_min: 85 }
      - axe_accessibility: { level: critical }
      - form_validate:
          selector: "form#contact"
          fields:
            email: "test@invalid"
          expect_errors: ["email"]
- if: ${qa.lighthouse.perf} < 85 or ${qa.form_validate.failed} > 0
  then:
    - focus_chat: {}
    - send_message:
        text: |
          Findings: perf=${qa.lighthouse.perf}, form errors=$
{qa.form_validate.failed}.
          Please optimize bundle and tighten email validation. Rebuild and
share preview.
    - await_response: { timeout: 600000 }
    - extract_preview_url: { save_as: preview }
    - qa_suite: { url: ${preview}, asserts: [ 'lighthouse>=85',
'form_validate_ok' ] }
```

24.5 Aksept & porter (gates)

- **Merge-gate:** PR kan ikke merges før alle kritiske QA-asserts er grønne.
- **Release-gate:** V1 krever grønn Lighthouse, 0 kritiske a11y-funn og bestått røyk-test.

24.6 Feilrapport (bug ticket mal)

```
Title: [QA] /contact email validation accepts invalid address
Steps: open ${preview}/contact → type `test@invalid` → submit
Expected: show validation error and prevent submit
Actual: form submits successfully
Evidence: screenshot, console log excerpt, network HAR
Severity: medium
Linked WO/PR: WO-…, PR-#…
```

24.7 Sporbarhet og læring

- Testresultater lenkes til **WO/PR** og lagres i §20 logg/arkiv.
- Selektor-drift oppdages (UI endret) → lag ny fallback-regel.
- Gjentatte funn → foreslå «sjekklistesnutt» i PR-malen.

---

25) QA-runner (C# skjelett)

For å støtte `qa_suite`-stegene på Windows, kan MadEasy AI Browser inkludere en innebygd **QA-runner**. Denne kjører tester som Lighthouse, axe-core og enkle formvalideringer via Playwright/Puppeteer.

25.1 Arkitektur

- **QAService** (C# .NET)
- Tar inn en `QAJob` (URL + asserts)
- Starter headless Chromium (Playwright)
- Kjører: Build-status, Lighthouse, axe-core, Formsjekk
- Returnerer `QAResult` JSON til Browser-kjerne
- **Bindings:** gRPC eller IPC for å motta jobber og sende resultater
- **Artefakter:** skjermbilder, HAR-filer, JSON-rapporter

25.2 C# kode (skjelett)

```
using System;
using System.Diagnostics;
using System.IO;
using System.Threading.Tasks;
using Microsoft.Playwright;

namespace MadEasy.QA
{
    public class QAJob
```

```
    {
        public string Url { get; set; } = string.Empty;
        public int PerfMin { get; set; } = 85;
        public string FormSelector { get; set; } = string.Empty;
    }

    public class QAResult
    {
        public bool BuildSuccess { get; set; }
        public int LighthouseScore { get; set; }
        public bool AccessibilityOk { get; set; }
        public bool FormValidationOk { get; set; }
        public string ScreenshotPath { get; set; } = string.Empty;
        public string ReportPath { get; set; } = string.Empty;
    }

    public class QAService
    {
        public async Task<QAResult> RunAsync(QAJob job)
        {
            var result = new QAResult();
            using var pw = await Playwright.CreateAsync();
            await using var browser = await pw.Chromium.LaunchAsync(new
BrowserTypeLaunchOptions { Headless = true });
            var context = await browser.NewContextAsync();
            var page = await context.NewPageAsync();

            // Navigate to preview URL
            await page.GotoAsync(job.Url);
            result.BuildSuccess = true; // simplify; hook into CI log parsing

            // Lighthouse run (via CLI, capture output)
            var lhReport = Path.GetTempFileName() + ".json";
            var psi = new ProcessStartInfo("lighthouse", $"{job.Url} --
output=json --output-path={lhReport} --quiet")
            {
                RedirectStandardOutput = true,
                RedirectStandardError = true,
                UseShellExecute = false
            };
            var proc = Process.Start(psi);
            proc.WaitForExit(60000);
            if (File.Exists(lhReport))
            {
                var json = File.ReadAllText(lhReport);
                // TODO: parse Lighthouse JSON for perf score
                result.LighthouseScore = 88;
                result.ReportPath = lhReport;
            }

            // Accessibility (axe-core inject)
```

```
            await page.AddScriptTagAsync(new PageAddScriptTagOptions { Path
= "axe.min.js" });
            var axeResult = await page.EvaluateAsync("await axe.run()") as
object;
            result.AccessibilityOk = axeResult != null; // TODO: parse
violations

            // Form validation test
            if (!string.IsNullOrEmpty(job.FormSelector))
            {
                var form = await page.QuerySelectorAsync(job.FormSelector);
                if (form != null)
                {
                    await page.FillAsync(job.FormSelector + "
input[type=email]", "test@invalid");
                    await page.ClickAsync(job.FormSelector + "
button[type=submit]");
                    // Check for validation message
                    var error = await
page.QuerySelectorAsync(".error, .validation-message");
                    result.FormValidationOk = error != null;
                }
            }

            // Screenshot
            var shotPath = Path.GetTempFileName() + ".png";
            await page.ScreenshotAsync(new PageScreenshotOptions { Path =
shotPath, FullPage = true });
            result.ScreenshotPath = shotPath;

            return result;
        }
    }
}
```

25.3 Bruk

- `QAService.RunAsync(QAJob)` kalles fra Executor når preview-URL er funnet.
- Resultat sendes tilbake som `QAResult` og lagres i Session/Events.
- Funn → auto-prompt i chat (til dev-AI) for forbedringer.

25.4 Utvidelser (V1)

- Parse faktisk Lighthouse JSON for alle metrikker (LCP, TBT, CLS).
- Parse axe-core violations → severity, node, fix.
- Eksport til PDF (QuestPDF) med rapport + skjermbilder.
- Visual regression: compare screenshot diffs.