BasicFakta

SaaS platform — product docs, architecture, deployment.

Overview

BasicFakta Overview

SaaS platform — product docs, architecture, deployment.

Owner: John **Last Verified:** 2026-02-17

Contents

To be populated from BasicFakta project documentation

Active Tasks

> Last Verified: 2026-02-17 | Owner: John

BasicFakta — Active Tasks

Open Tasks

No currently active BasicFakta tasks in Mission Control.

Blocked Tasks

- **#100** — Improve fake news detection accuracy (BLOCKED)
  - **Reason:** Backend API code not in repo (hosted separately on basicfakta.no)
  - **Cannot proceed:** Algorithm source code not accessible
  - **Dependency:** Backend import required
  - **Test status:** 23/65 tests fail, factScore > 30 for known fake news
  - **Resolution:** Need backend source code from Alem

Recently Completed

- **13-item audit** — Security, accessibility, UX improvements (commit ce800c2)
- **Deployment** — Vercel auto-deploy configured
- **GitHub auth fix** — Switched from SSH to HTTPS (gh CLI token)

Planned Work

- **Backend integration** — Import backend source code to repo
- **Algorithm improvements** — Once backend is accessible
- **Test coverage** — Improve from 42/65 passing to 65/65
- **Accuracy tuning** — Reduce false positives for known fake news

Key Decisions

> Last Verified: 2026-02-17 | Owner: John

BasicFakta — Key Decisions

Strategic Decisions

Backend Separation (Date TBD)

Decision: Host backend API separately from frontend repo. **Rationale:** Protects proprietary algorithm, easier to scale backend independently. **Trade-off:** Makes algorithm improvements harder, blocks accuracy work. **Current impact:** Task #100 blocked pending backend import.

Deployment on Vercel (Date TBD)

Decision: Auto-deploy from GitHub main branch via Vercel. **Rationale:** Zero-config deployment, global CDN, preview deployments. **Status:** Active and working.

Technical Decisions

Frontend Stack

- Modern web framework (details in repo)
- Vercel hosting
- GitHub version control (HTTPS auth)

Security Measures (2026-02)

Decision: Implement comprehensive security hardening. **Implemented:**

- Input sanitization (XSS prevention)

- Rate limiting (authentication bypass fix)
- API key sanitization
- Security headers via vercel.json

Accessibility (2026-02)

Decision: Achieve WCAG 2.1 compliance. **Implemented:**

- Screen reader support
- Keyboard navigation
- Mobile-friendly touch targets

Development Decisions

GitHub Auth (2026-02)

Decision: Switch from SSH to HTTPS authentication. **Reason:** ~/.zshenv had GITHUB_TOKEN=CHANGE_ME placeholder that overrode gh CLI token. **Fix:** Commented out line 40, use gh CLI token (ghp_ PAT in ~/.config/gh/hosts.yml).

Test Strategy (Current)

Status: 42/65 tests passing **Blocker:** 23 tests fail due to backend algorithm issues (factScore too high for known fake news) **Next step:** Backend import required before test fixes

Operational Decisions

Email Domain

- **Domain:** basicfakta.no
- **Status:** Email monitoring blocked (task #79)
- **Issue:** Missing IMAP credentials (host/port/user)
- **Required from Alem:** IMAP access details

Project Overview

> Last Verified: 2026-02-17 | Owner: John

BasicFakta — Project Overview

What is BasicFakta?

BasicFakta (basicfakta.no) is a SaaS fact-checking tool that analyzes news articles and claims to detect misinformation. It uses AI-powered algorithms to generate "fact scores" indicating the likelihood that content is fake news.

Current Status

- **Phase:** Deployed (live at basicfakta.no)
- **Type:** Internal product / SaaS offering
- **Backend:** External hosted API (not in main repo)
- **Frontend:** Repository available

Features

1. **Fact Checking** — Analyze news articles and claims
2. **Fact Scoring** — 0-100 score (lower = more likely fake)
3. **URL Extraction** — Extract and analyze URLs from articles
4. **Share Results** — Share fact-check results
5. **Mobile Support** — Responsive design

Tech Stack

- **Frontend:** Modern web framework
- **Backend API:** Hosted separately at basicfakta.no
- **Algorithm:** Proprietary fake news detection (not open source)

Recent Work

Audit & Fixes (2026-02)

13 audit items fixed and deployed (commit ce800c2):

Critical Fixes

- **url-extractor.js created** — Was missing, causing extraction failures
- **XSS vulnerability** — Input sanitization implemented
- **Rate limit bypass** — Fixed authentication bypass
- **API key sanitization** — Sensitive data protection

Medium Improvements

- **Security headers** — Added via vercel.json
- **Email validation** — Enhanced validation
- **SEO meta tags** — Improved search visibility
- **WCAG accessibility** — Screen reader support, keyboard navigation
- **Mobile UX** — Touch-friendly interface
- **Share results** — Social sharing functionality
- **Loading states** — Improved user feedback

Known Issues

Backend Separation

- **Backend code not in repo** — Hosted separately, not accessible for improvements
- **Algorithm improvements blocked** — Cannot improve detection accuracy without backend access
- **Test failures** — 23/65 tests fail because factScore > 30 for known fake news
- **Resolution needed** — Backend must be imported or accessed before further accuracy work

Key Documents

- **Repository:** GitHub (switched from SSH to HTTPS auth)
- **Deployment:** Vercel auto-deploy from main branch
- **Audit Report:** ~/system/reports/ (details TBD)

Specifications Index

> Last Verified: 2026-02-17 | Owner: John

BasicFakta — Specifications Index

Frontend Code

- **Repository:** GitHub (HTTPS auth, gh CLI token)
- **Main branch:** Auto-deploys to basicfakta.no via Vercel
- **Framework:** Modern web stack (details in repo)

Backend API

- **Hosting:** Separate external API at basicfakta.no
- **Source code:** Not in repo (blocks accuracy improvements)
- **Algorithm:** Proprietary fake news detection
- **Status:** Access required for further development

Security Specifications

Implemented (2026-02)

- XSS prevention via input sanitization
- Rate limiting with authentication
- API key sanitization
- Security headers (vercel.json)

Pending

- Backend security audit (requires backend access)
- Penetration testing (scheduled TBD)

Accessibility Specifications

WCAG 2.1 Compliance

- Screen reader support
- Keyboard navigation
- Mobile touch targets (44x44px minimum)
- Color contrast ratios
- Loading state announcements

Testing Specifications

Current Status

- **42/65 tests passing** (64.6% pass rate)
- **23 tests failing** — factScore > 30 for known fake news sources

Test Categories

1. URL extraction tests
2. Fact scoring accuracy tests
3. XSS prevention tests
4. Rate limiting tests
5. API integration tests

Blockers

- Backend algorithm not accessible
- Cannot tune scoring thresholds
- Cannot improve detection accuracy

Audit Reports

2026-02 Audit (13 items)

Critical:

- url-extractor.js (CREATED)
- XSS fix (FIXED)
- Rate limit bypass (FIXED)
- API key sanitization (FIXED)

Medium:

- Security headers (ADDED)
- Email validation (IMPROVED)
- SEO meta tags (ADDED)
- WCAG accessibility (IMPLEMENTED)
- Mobile UX (IMPROVED)
- Share results (ADDED)
- Loading states (IMPROVED)

Deployment: Commit ce800c2, Vercel auto-deployed

Dependencies

Required from Alem

1. **Backend source code** — For algorithm improvements (task #100)
2. **IMAP credentials** — For email monitoring (task #79)
3. **Test data** — Known fake news URLs for accuracy testing

External Services

- Vercel (hosting)
- GitHub (version control)
- basicfakta.no API (backend)

Code Quality Audit 2026-07-08 (MC #105042)

Proveo Code Quality Audit — basicfakta

MC #105042 | 2026-07-08 | Audit type: read-only (no code mutations, no deploy, no commits)

Project: /Users/makinja/projects/internal/basicfakta Note: BUILD-BLUEPRINT.md/CLAUDE.md describe this as a plain Node.js/Express serverless app (Vercel functions in `api/`, vanilla JS in `src/`), NOT the React/Next.js/TS stack implied in the dispatch description. Tests are hand-rolled Node scripts (no Jest), so "coverage" comes from `c8` (Istanbul), not a framework-native coverage command.

Verdict: CONCERNS

No blocking defects — 0 lint errors, 0 type errors, all 139 tests pass. "CONCERNS" reflects real coverage gaps in security/data-layer code and a large amount of high-complexity, hard-to-maintain logic (esp. one 36-complexity, 270-line function) that should be refactored before this codebase grows further.

---

1. Lint

Command: `npm run lint` (`eslint src tests`) Result: **0 errors, 35 warnings**, exit code 0. Full output: `lint-output.txt`

Breakdown by rule:

Rule	Count
`quotes` (must use singlequote)	25 — all in `tests/run-manual-tests.js` (orphaned file, see §3)
`no-unused-vars`	7 (analyzer.js x2, heuristics.js, security.js, server.js, ssb-verifier.js, heuristics.test.js, unit-deep.test.js)

Rule	Count
`no-useless-escape`	2 (heuristics.js regex patterns)

No errors anywhere. All fixable via `--fix` (eslint reports 25 auto-fixable).

2. Type check / Complexity

Command: `npm run typecheck` (`tsc --noEmit`) Result: clean, exit code 0, no output. Full output: `typecheck-output.txt`

Caveat (material): `tsconfig.json` has `"checkJs": false` and `"strict": false`, and `include` is scoped to `src/**/*.js` only (excludes `api/`, `tests/`). With `checkJs: false`, tsc does NOT type-check JS file bodies — this run only confirms the files parse as valid syntax. It is not meaningful type-safety verification. If real type coverage is wanted, `checkJs: true` needs enabling (out of scope for this read-only audit — flagging as a finding, not fixing).

Complexity (measured via one-off `npx eslint --rule '{"complexity":["warn",10],...}'` override, read-only, no config file changed). Full output: `complexity-output.txt`.

Top 10 most complex functions:

#	Function	File:line	Complexity	Lines
1	`analyzeHeuristics`	src/heuristics.js:507	**36**	270
2	`combineResults`	src/analyzer.js:576	**29**	120
3	`analyzeText`	src/analyzer.js:231	23	151
4	`getStats`	src/database.js:72	23	—
5	`fetchTableData`	src/ssb-verifier.js:240	22	—
6	`isPrivateIP`	src/url-extractor.js:28	18	—
7	`extractWebPageContent`	src/url-extractor.js:272	17	—
8	`extractYouTubeContent`	src/url-extractor.js:138	16	—
9	`detectLanguage`	src/language-detector.js:58	15	—
10	`runAllTests`	tests/run-tests.js:66	14	— (orphaned file, §3)

Also flagged: `parseAIResponse` (analyzer.js:505, complexity 11), an anonymous arrow fn (analyzer.js:519, complexity 19), `verifyClaim` (ssb-verifier.js:303, complexity 12), `validateURL` (url-extractor.js:63, complexity 12), one 5-deep nested block (ssb-verifier.js:267, max-depth 5 vs limit 4).

Largest files by LOC: heuristics.js (847), analyzer.js (717), ssb-verifier.js (500), url-extractor.js (381), middleware/security.js (372).

`analyzeHeuristics`, `combineResults`, and `analyzeText` are the three core pipeline functions and all exceed complexity 20 — these are the priority refactor candidates (see fix list).

3. Dead code detection

Ran both `ts-prune` and `knip --no-config-hints` (one-off npx, no devDependency added to repo). Full outputs: `ts-prune-output.txt`, `knip-output.txt` (also captured in `deadcode-output.txt`).

ts-prune (scoped to tsconfig's `src/**/*.js` include, so misses `api/` and `tests/`): 10 items reported, but all are "(used in module)" — i.e. exported but only consumed inside their own file. These are export-hygiene items (could be de-exported), not actual dead code.

knip (wider scan, more useful signal here) reported:

- **Unused files (7):** `api/analyze.js`, `api/feedback.js`, `api/health.js`, `src/sentry.js`, `src/url-extractor.js`, `tests/run-manual-tests.js`, `tests/run-tests.js`
  - **False positives, verified by grep:** the 3 `api/*.js` files are Vercel serverless entry points invoked by the platform router (per `vercel.json` rewrites), not imported by other modules — knip has no visibility into that root. `src/sentry.js` and `src/url-extractor.js` are both required by `api/analyze.js` (confirmed: `grep -rn "require.*sentry" api/analyze.js` and `grep -rln "require.*url-extractor" api` both hit) — knip misses this because it doesn't trace `api/` as an entry root either.
  - **Genuine findings:** `tests/run-manual-tests.js` and `tests/run-tests.js` are NOT referenced by any `package.json` script, CI workflow, or other source file (grep confirmed empty). Superseded by `tests/unit-deep.test.js` / `tests/heuristics.test.js` per naming and content overlap. Safe deletion candidates.
- **Unused dependency (1):** `@sentry/node` in package.json — flagged unused by knip's dependency graph, but this is also a false positive: `src/sentry.js` (line uses it) is itself only reached via the untraced `api/` root, so the whole chain is invisible to knip. Do not remove.
- **Unused exports (27):** concentrated in `src/middleware/security.js` (11 exports), `src/rate-limiter.js` (5), `src/ssb-verifier.js` (5), `src/database.js` (4), `src/analyzer.js` (2). Same caveat — many of these ARE used by `api/*.js` files knip can't see as roots. Not verified individually line-by-line (out of timebox); treat as "needs manual triage," not "confirmed dead."

Net dead-code finding with highest confidence: 2 orphaned test files (`tests/run-manual-tests.js`, `tests/run-tests.js`, ~11KB combined) plus their 25 lint warnings — both the code and its lint noise disappear if deleted.

4. Test coverage

Three test suites, all passing:

Suite	Command	Result
Heuristics	`npm test`	50/50 passed
Unit-deep	`npm run test:unit`	62/62 passed
API integration	`npm run test:api`	27/27 passed
Total		**139/139 passed, 0 failures**

e2e (`npm run test:e2e`, Playwright) skipped per timebox instruction — unit+integration coverage sufficient for this audit; explicitly noting the skip, not silently omitting it.

Coverage (`coverage/coverage-summary.json`, file mtime 2026-07-01 — regenerated fresh during this audit via `npx c8` over `heuristics.test.js` + `unit-deep.test.js` combined, and the numbers reproduced **exactly** match the existing file, so the "staleness" is cosmetic — the underlying source hasn't changed since Jul 1 in a way that would move these numbers):

Metric	%
Statements	66.94
Lines	66.94
Branches	84.54
Functions	61.11

Per-file (only files touched by these two suites appear in the report — `api/*.js`, `src/server.js`, `src/database.js`, `src/emailer.js`, `src/sentry.js` are **0% / not instrumented at all**, since `api-integration.test.js` tests API *logic* without importing the handler modules under c8):

10 worst-covered files (lines %, lowest first; only instrumented files listed):

File	Lines %	Functions %	Branches %
`src/analyzer.js`	45.18	44.44	23.8
`src/ssb-verifier.js`	49.8	33.33	85.71
`src/middleware/security.js`	46.77	38.46	89.28
`src/heuristics.js`	94.33	100	93.27
`src/rate-limiter.js`	92.89	100	85.71
`src/language-detector.js`	97.05	100	87.5

File	Lines %	Functions %	Branches %
Not instrumented (0%, no coverage data): `api/analyze.js`, `api/feedback.js`, `api/health.js`, `src/server.js`, `src/database.js`, `src/emailer.js`, `src/sentry.js`, `src/url-extractor.js`	—	—	—

`analyzer.js`, `ssb-verifier.js`, and `middleware/security.js` are the three weakest-covered files that DO have partial coverage — and two of them (`analyzer.js`, `security.js`) also contain the highest-complexity functions found in §2. High complexity + low coverage is the same code twice.

Top 5 Prioritized Issues

1. **`analyzer.js` combines the highest complexity in the codebase with the lowest coverage.** `combineResults` (complexity 29) and `analyzeText` (complexity 23, the main pipeline entry) sit in a file at only 45.18% line / 23.8% branch coverage. This is the analysis pipeline's core — the least-tested, hardest-to-reason-about code is exactly where a regression would be most costly. Priority: add branch-level tests for the AI/heuristic-fallback paths, then refactor `combineResults` and `analyzeText` to extract sub-functions (target complexity <15 each).
2. **`analyzeHeuristics` (heuristics.js:507) has complexity 36 across 270 lines** — the single highest-complexity function found. Coverage is good (94.33%) so behavior is protected, but maintainability is poor: any future pattern addition risks unintended interaction with existing branches. Recommend decomposing into per-category scoring functions before adding more patterns.
3. **`security.js` (auth/input-validation middleware) is only 46.77% line-covered and 38.46% function-covered**, with 11 flagged (possibly-unused, needs manual triage) exports. Security middleware being the least-tested layer is a QA risk independent of the dead-code ambiguity — recommend a dedicated test pass on `sanitize`, `validateAnalyzeInput`, `errorHandler`, and `corsConfig` before touching complexity/dead-code cleanup here.
4. **`typecheck` is a no-op safety net.** `checkJs: false` + `strict: false` means `tsc --noEmit` only validates syntax, not types — "0 type errors" in this report should not be read as "no type-related bugs." If type-safety is actually wanted here, enabling `checkJs: true` (as an incremental, separately-scoped change) would surface real signal; right now the script name is misleading about what it verifies.
5. **2 orphaned test files** (`tests/run-manual-tests.js`, `tests/run-tests.js`) contribute 25 of the 35 total lint warnings (all `quotes`) and are not wired into any script/CI. Low-risk, low-effort cleanup: delete both files, re-run lint to confirm warning count drops from 35→~9.

(Note: `api/*.js`, `src/sentry.js`, `src/url-extractor.js`, and the 27 "unused exports" flagged by knip are NOT safe to delete — verified as false positives/needing manual triage, see §3.)

Evidence files (all under ~/system/evidence/105042/)

- `audit-report.md` — this file
- `lint-output.txt` — full `npm run lint` output
- `typecheck-output.txt` — full `npm run typecheck` output
- `complexity-output.txt` — eslint with complexity/max-depth/max-lines-per-function rules
- `deadcode-output.txt`, `ts-prune-output.txt`, `knip-output.txt` — dead code detection raw outputs
- `coverage-summary.txt` — copy of project's `coverage/coverage-summary.json`
- `coverage-run-output.txt` — fresh c8 run, unit-deep suite only
- `coverage-combined-output.txt` — fresh c8 run, heuristics + unit-deep combined (reproduces the committed coverage-summary.json exactly)
- `test-unit-run-output.txt`, `test-heuristics-output.txt`, `test-api-output.txt` — full test logs for all 3 suites (139/139 passed)