Product Requirements Document (PRD) — Story Scope

**1) Overview**

* StoryScope is a gamified story estimation tool that lets users submit feature/user stories and get consistent complexity estimates, track progress, and export/import to Jira.
* Supports authenticated multi-user workflows, admin controls, and exportable reports.

**2) Goals**

* Provide fast, consistent estimates for stories with minimal friction.
* Motivate users via streaks, levels, achievements.
* Enable project tracking via dashboard stats and CSV/Jira integration.
* Operate with low operational overhead and easy deployment (Railway).

**3) Non-Goals**

* Full agile lifecycle management (sprints/boards).
* Deep NLP accuracy benchmark; current approach uses a mock/simple scoring heuristic unless an external NLP is configured.
* Real-time collaboration (presence).

**4) Target Personas**

* Product Owner/BA: drafts stories, needs quick estimates and exports to Jira.
* Developer/Tech Lead: refines description/labels and validates complexity trends.
* Engineering Manager: reviews dashboard stats, exports reports, manages admins.

**5) Key User Stories**

* As a user, I can sign up, log in, and create stories with summary/description/labels to get a complexity score.
* As a user, I can review recent stories, search, paginate, and view details.
* As a user, I see dashboard stats (total, average complexity, latest), streak/level/XP, and achievements; confetti for milestone unlocks.
* As a user, I can export a dashboard report CSV and Jira-friendly CSV; import a CSV with a preview.
* As an admin, I can view users and manage roles (promote/demote).

**6) Core Flows**

* Auth: email/password → JWT stored in localStorage → protected routes.
* Estimate: POST story → response includes ID and complexity → UI toasts + optional confetti on milestone.
* Browse: GET stories with search/pagination → recent list → detail view → edit/delete.
* Dashboard: GET stats → shows metrics, streak, level, achievements → download report.
* Jira CSV: GET template; POST export → download CSV; Import → preview (mapping is simple, future: map columns).

**7) Functional Requirements**

* Authentication
  + Sign up, login, me endpoints; JWT-based; role in token (user/admin).
  + Admin-only endpoints protected.
* Story Management
  + Create with summary, description, labels[].
  + List with search, pagination; view detail; edit; delete.
  + All endpoints scoped to the authenticated user.
* Analytics & Gamification
  + Stats: total stories, average complexity, latest.
  + Level = floor(total/5), streak = simple heuristic (draft); achievements: first 5, week streak, high complexity.
  + Milestone-only confetti and toasts.
* Jira Integration (CSV Phase)
  + Download Jira template.
  + Export stories (Summary, Description, Labels, Story Points).
  + Import CSV with preview (no persistence yet).
* Admin
  + List users; promote/demote roles.
* Reporting
  + Dashboard user-scoped report CSV with totals, average, and latest N records.

**8) Non-Functional Requirements**

* Usability: modern, responsive, accessible dark UI; keyboard navigation; clear error states.
* Performance: list pagination; DB indexes on user\_id, created\_at; search on summary/description.
* Security: JWT signed; password hashing (bcrypt); CORS limited; Helmet (future).
* Reliability: graceful error messages; dev proxy for local; SQLite fallback locally, Postgres in prod.
* Observability: server logs, health endpoint; easily add Sentry later.

**9) Success Metrics**

* Time-to-estimate: < 5 seconds average from form submit to response.
* Adoption: number of active users per week, number of estimates/story.
* Engagement: maintained streaks; achievements unlocked; report downloads.
* Export usage: count of Jira CSV exports.

**10) Release Plan (Phases)**

* Phase 1 (MVP): Auth, submit/list/detail, stats, CSV export/template, basic achievements, report CSV.
* Phase 2: Responsive polish, DB indexes, error handler, retries/timeouts, admin role UI, import mapping.
* Phase 3: Jira OAuth and project mapping; push on submit.

High-Level Design (HLD) - Story Scope

**A) Architecture Overview**

* Frontend: React + Vite (TypeScript). Protected routing, API helper to attach JWT, modern dark UI.
* Backend: Node.js (Express), Postgres via Knex. SQLite fallback for local.
* Deployment: Single container image (Docker) building the web UI and serving via the API. Railway.
* Optional NLP: Mock scoring built-in; can switch to external NLP service via NLP\_URL.

**B) Components**

* Web (apps/web)
  + Router: /auth, protected /(dashboard|submit|recent|story/:id|settings|admin).
  + Pages:
    - Dashboard: metrics, streak/level/XP, achievements, report download.
    - Submit: form with toast; milestone-only confetti.
    - Recent: search, pagination, list; bulk actions (delete).
    - StoryDetail: inline edit (summary/description/labels), delete.
    - Settings: API override, feature toggles, download Jira template/export, import CSV preview.
    - Auth: signup/login forms; token saved in localStorage.
    - Admin: user list and role controls (promote/demote).
  + Lib:
    - apiFetch attaches Bearer token; respects Settings override.
    - auth helpers for session and role.
* API (apps/api)
  + Auth
    - POST /auth/signup, /auth/login
    - GET /auth/me
    - JWT with role claim; bcrypt hashing.
  + Stories (user-scoped)
    - POST /estimate (requires auth): computes complexity (mock or external NLP).
    - GET /stories (search, pagination), GET /stories/:id.
    - PATCH /stories/:id, DELETE /stories/:id.
  + Stats & Reports
    - GET /stats (auth).
    - GET /report.csv (auth) — dashboard CSV.
  + Jira CSV
    - GET /jira/template.csv — Jira template CSV.
    - POST /jira/export (auth) — export user’s stories.
  + Admin
    - GET /admin/users (requireAdmin).

**C) Data Model (Postgres)**

* users
  + id (int PK), email (unique), password\_hash (text), role (text: 'user'|'admin', default 'user'), created\_at (timestamp).
* stories
  + id (int PK), user\_id (FK [users.id](http://users.id/)), summary (string, not null), description (text), labels (text JSON), complexity\_score (int), created\_at, updated\_at.

Indexes recommended:

* stories(user\_id, created\_at desc)
* stories(summary text\_pattern\_ops) & stories(description text\_pattern\_ops) or use ILIKE with trigram extension (future).
* users(email unique)

**D) API Contracts (selected)**

* Auth
  + POST /auth/signup → { token, user: { id, email, role } }
  + POST /auth/login → { token, user }
  + GET /auth/me → { user }
* Stories
  + POST /estimate body: { summary, description, labels[] } → { id, ... }
  + GET /stories?search=&page=&limit= → { items: Story[], total, page, limit }
  + GET /stories/:id → Story
  + PATCH /stories/:id body: partial fields → Story
  + DELETE /stories/:id → { ok: true }
* Stats/Reports
  + GET /stats → { total, average\_complexity, latest[] }
  + GET /report.csv → CSV
* Jira CSV
  + GET /jira/template.csv → CSV header
  + POST /jira/export → CSV body

All protected endpoints require Authorization: Bearer <token>.

**E) Security**

* JWT with HS256, JWT\_SECRET set in prod. Expiry 7d.
* Passwords hashed with bcryptjs, 10 rounds.
* CORS: allow only FE origin in prod (tighten from \*).
* Future: Helmet, rate limiting on auth endpoints.

**F) Error Handling & Resilience**

* Server
  + Consistent JSON errors: { error: string, details? }
  + Graceful DB init; health endpoint /health.
* Client
  + apiFetch throws on non-OK; pages catch and show friendly messages.
  + Search is debounced; network errors render recoverable guidance.
* Deploy
  + OptionalDependencies for sqlite3 to avoid Alpine build fails.
  + Root Dockerfile builds web and runs API; SPA fallback guarded.

**G) Configuration**

* Env Vars (Railway)
  + DATABASE\_URL (Postgres)
  + PGSSLMODE=require (or DATABASE\_SSL=true)
  + JWT\_SECRET (prod secret)
  + NODE\_ENV=production
  + NLP\_URL (optional external NLP)
* Frontend
  + Dev proxy: /auth, /stories, /estimate, /stats, /jira, /admin, /report.csv to <http://localhost:8000.>
  + VITE\_API\_URL optional; Settings overrides per-browser.

**H) Deployment & CI**

* Railway using root Dockerfile:
  + npm ci --prefix apps/web && npm run build --prefix apps/web
  + npm ci --omit=dev --prefix apps/api
  + CMD node server.cjs
* Clear build cache if build context changed or deps updated.

**I) Observability**

* Logs: request/response errors logged to stdout (Railway Logs).
* Health: /health returns { status: "ok" }.
* Future: Sentry, metrics, structured logs.

**J) Scalability & Future Work**

* Move search to trigram indexes or dedicated search.
* Cursor-based pagination.
* Background NLP scoring queue (BullMQ/Redis) if heavy NLP.
* Jira OAuth app + push on submit; field mapping UI.

**K) Risks and Mitigations**

* Build failures (native deps): moved sqlite3 to optional; use Postgres in prod.
* DNS/proxy issues: documented checks, proxy off, DNS reset.
* Security gaps: plan for CORS tightening and Helmet; enforce password rules.
* Data growth: add indexes and pagination; report limits.

**L) Acceptance Criteria (MVP)**

* Users can sign up, login, submit stories and view results with complexity.
* Dashboard shows totals/avg/latest; Download Report works.
* Recent supports search/pagination; Detail supports edit/delete.
* Jira template download and CSV export works; import preview displays.
* Admin page lists users, role management functional.
* Auth-protected routes enforced on both client and server.
* Railway deploy succeeds and app runs via public URL.

If you want this as markdown files in the repo (docs/PRD.md, docs/HLD.md), I can add them next.