Project Working & Milestone Achievement Guide (Milestones 1–5)

This document explains how the FastAPI backend is structured, how to use it, and how the repository implements Milestones 1–5 from the assignment.

> Scope note: This repo focuses on Milestones 1–5 (Foundation → Spec step). Milestone 6+ (code generation/validation, full LangGraph checkpointing, Langfuse traces, etc.) is not implemented here.

---

1) What this system does (high level)

The backend supports a product-planning pipeline:

1. **Authenticate** (User/Admin via JWT)
2. **Create a Project** containing a free-text Product Request
3. **(Optional) Upload a supporting PDF** stored on the filesystem
4. **Start a "backlog generation run"** which performs mandatory web research (Tavily)
5. **Generate Epics** grounded in that research + persist a Mermaid dependency graph
6. **Approve Epics** (approval gate)
7. **Generate Stories** for an approved epic + approve them (approval gate)
8. **Generate a Spec** for an approved story (spec-first), then **approve/reject** the spec

Real-time progress is modeled as **Run Events** that are persisted to the database and can be streamed through WebSockets.

---

2) Repository architecture (how the code is organized)

Core building blocks

- **FastAPI app + router wiring**: `app/main.py`
- **Configuration** (env vars, storage root, DB URL, JWT, API keys): `app/core/config.py`
- **Database models** (SQLite via SQLAlchemy): `app/db/models.py`
- **Auth + role enforcement**: `app/api/routers/auth.py`, `app/api/deps.py`
- **Project and file upload**: `app/api/routers/projects.py`, `app/services/storage.py`
- **Runs + Research job**: `app/api/routers/runs.py`, `app/services/research.py`
- **Run events (persistence + publish)**: `app/services/run_events.py`, REST views in `app/api/routers/run_events.py`
- **Epic generation**: `app/api/routers/epics.py`, `app/services/epic_generation.py`
- **Story generation**: `app/api/routers/stories.py`, `app/services/story_generation.py`
- **Spec generation & approval (Milestone 5)**: `app/services/spec_generation.py` + WebSocket control in `app/api/routers/ws.py`

Key data entities

Stored in SQLite (default `./data/app.db` unless overridden):

- `User` (role: `user` or `admin`)
- `Project` (owner + product_request)
- `Artifact` (uploaded PDFs)
- `Run` (execution instance: research/epics/stories/specs)
- `RunEvent` (auditable event stream per run)
- `ResearchAppendix` (persisted research results per backlog run)
- `EpicBatch` + `Epic` (with approval)
- `StoryBatch` + `Story` (with approval)
- `SpecDocument` (versioned specs with approve/reject)

---

3) Storage layout (filesystem)

The backend stores some artifacts on disk under `STORAGE_ROOT` (defaults to `data/`).

Typical structure:

- `data/projects/<project_id>/uploads/<uuid>.pdf`
  Supporting PDF uploads.
- `data/projects/<project_id>/runs/<run_id>/research.md`
  Persisted Research Appendix markdown.
- `data/projects/<project_id>/runs/<run_id>/epic_dependency_graph.mmd`
  Mermaid epic dependency graph.

Database rows keep references to these artifacts via relative paths.

---

4) How to run locally

Install

Use your virtual environment and install dependencies:

```
python -m pip install -r requirements.txt
```

Configure

Copy `.env.example` → `.env` and set at minimum:

- `JWT_SECRET` (required for secure auth)
- `TAVILY_API_KEY` (required for Milestone 2 backlog runs)

Optional:

- `SEED_ADMIN_EMAIL` / `SEED_ADMIN_PASSWORD` to create an admin account at startup
- `OPENAI_API_KEY` to use OpenAI for epics/stories/specs (otherwise the system uses deterministic fallbacks)

Start the server

```
python -m uvicorn app.main:app --reload
```

Then open Swagger UI:

- `http://127.0.0.1:8000/docs`

---

5) Milestone 1: Foundation (“User Can Start”)

What was implemented

Authentication & onboarding

- **Signup**: `POST /auth/signup`
- **Login**: `POST /auth/login` (OAuth2 password form)
- JWT is returned as `access_token`; use it as:

```
Authorization: Bearer <token>
```

Role-based access

- Users can only see/modify **their own** projects, runs, and artifacts.
- Admin-only endpoints exist under `/admin/*`.

Project initiation

- **Create project**: `POST /projects` with `{ "product_request": "..." }`
- **List my projects**: `GET /projects`
- **Get project details + artifacts**: `GET /projects/{project_id}`

Supporting document upload (PDF)

- Upload: `POST /projects/{project_id}/documents` (multipart form)
- Stored as an `Artifact` row + a file under `data/projects/<project_id>/uploads/`.

Invalid input handling (examples)

- Empty Product Request → HTTP 400 (`Product Request cannot be empty`)
- Non-PDF upload → HTTP 400 (`Only PDF uploads are supported`)
- Corrupted PDF → HTTP 400 (`Unsupported or corrupted PDF document`)
- Size limit → HTTP 400 (`File too large (max <MAX_UPLOAD_MB>MB)`)

Demo scenario

1. Sign up → 2) Login → 3) Create project → 4) (Optional) upload PDF

At this point Milestone 1 is complete (a user can start and a project exists).

---

6) Milestone 2: Research (“System Uses Web + Market Evidence”)

What was implemented

Backlog generation run triggers web research

- Start backlog/research: `POST /projects/{project_id}/runs/backlog`
- This endpoint requires `TAVILY_API_KEY`.

Research is mandatory and persisted

- The background job calls Tavily search with multiple queries derived from the product request.
- It writes a `research.md` file under `data/projects/<project_id>/runs/<run_id>/research.md`.
- It persists a `ResearchAppendix` row containing:
  - consulted URLs (`urls_json`)
  - summary
  - “impact” explanation (why research matters downstream)

Users can view research

- List run events: `GET /runs/{run_id}/events`
- Fetch research appendix (DB + markdown): `GET /runs/{run_id}/research`

Run event observability (typical events)

During Milestone 2 you'll see events like:

- `run.started` – "Backlog Generation Started"
- `research.started` – "Research in progress"
- `research.completed` – "Research complete" + payload `{ url_count: ... }`

If the API key is missing, the backlog endpoint returns HTTP 400 with a clear message.

---

7) Milestone 3: Epic Generation

What was implemented

Research-first gate

Epic generation requires a prior research appendix for the project. If research doesn't exist, the server returns a clear HTTP 400 error.

Generate epics

- `POST /projects/{project_id}/epics/generate`
- Body supports:
  - `count` (bounded, 1–12)
  - `constraints` (free text; e.g., "must support SSO")

What an epic contains

Each Epic includes required Milestone 3 fields:

- title, goal
- in-scope / out-of-scope
- priority (P0/P1/P2) + reasoning
- dependencies (as epic titles)
- risks, assumptions, open questions

- success metrics

Mermaid dependency graph artifact

- Mermaid is produced and persisted to:
  - `data/projects/<project_id>/runs/<run_id>/epic_dependency_graph.mmd`
- The API response also includes the Mermaid string.

Epic approval loop

- Approve batch: `POST /projects/{project_id}/epics/{batch_id}/approve`
- Approved epics become eligible for story generation (Milestone 4 gate).

LLM vs deterministic fallback

Epic generation uses OpenAI if `OPENAI_API_KEY` is configured; otherwise it uses a deterministic heuristic generator so the flow remains usable without an LLM key.

---

8) Milestone 4: User Story Generation ("Epics → Stories with Acceptance Criteria")

What was implemented

Approval gate

Story generation requires an **approved epic**.

Generate stories for a chosen epic

- `POST /projects/{project_id}/stories/generate`
- Request body includes:
  - `epic_id`
  - optional `constraints`
  - `count`

Story content includes

- User story statement
- Acceptance criteria (Given/When/Then list)
- Edge cases & failure modes
- Non-functional requirements
- Estimate (t-shirt size) + rationale
- Dependencies

Review & approval

- Approve batch: `POST /projects/{project_id}/stories/{batch_id}/approve`
- List latest story batch for an epic:
  - `GET /projects/{project_id}/stories?epic_id=<epic_id>`

Like Milestone 3, story generation can use OpenAI (if configured) or deterministic fallbacks.

---

9) Milestone 5: Spec-Driven Development (“Spec First, Then Code”)

What was implemented

This repo implements Milestone 5 primarily via **WebSockets** (real-time control), not via REST endpoints.

WebSocket endpoint

- `WS /ws/projects/{project_id}/specs?token=<jwt>`

The WebSocket authenticates via the JWT token in the query string and verifies the user owns the project.

Commands supported

Send JSON messages over the socket:

- Generate spec:
  - `{ "type": "specs.generate", "story_id": "...", "constraints": "..." }`
- Regenerate spec (with feedback):
  - `{ "type": "specs.regenerate", "story_id": "...", "constraints": "...", "feedback": "..." }`
- Get latest spec for a story:
  - `{ "type": "specs.get", "story_id": "..." }`
- Approve spec:
  - `{ "type": "specs.approve", "spec_id": "..." }`
- Reject spec:
  - `{ "type": "specs.reject", "spec_id": "...", "feedback": "need rate limiting" }`

What a spec contains

The stored `SpecDocument` includes (Milestone 5 requirements):

- Overview + goals
- Functional requirements mapped to acceptance criteria
- API contracts (placeholder list)
- Data model changes (placeholder list)
- Security considerations
- Error handling strategy
- Observability plan
- Test plan mapping AC → test cases
- Implementation plan (files/modules to change)
- Mermaid diagrams:
  - `mermaid_sequence` contains a `sequenceDiagram`
  - `mermaid_er` contains an `erDiagram`

Approval gate

Specs start as `proposed` and must be explicitly **approved** via `specs.approve`. Specs can also be rejected with feedback via `specs.reject`.

> Note: Milestone 6 (code generation from approved spec) is not implemented in this repository.

---

10) Observability: Run Events (REST + WebSocket)

Persisted run events

Every run writes `RunEvent` rows. These are queryable via:

- `GET /runs/{run_id}/events`

Real-time streaming

The WebSocket routes support attaching to a run and receiving a live feed of events:

- `{ "type": "runs.attach", "run_id": "..." }`

Events are published through an in-process broker and delivered to subscribed WebSocket clients.

---

11) Admin operations (role-based access)

If you seed an admin via `.env` (`SEED_ADMIN_EMAIL` / `SEED_ADMIN_PASSWORD`), you can use:

- `GET /admin/users` – list users (admin only)
- `POST /admin/users/{user_id}/promote` – make user admin
- `POST /admin/users/{user_id}/demote` – remove admin role
- `DELETE /admin/users/{user_id}` – delete user

Safety rules are enforced:

- You cannot demote or delete yourself.
- The last remaining admin cannot be removed.

---

12) Configuration reference

All configuration is in `app/core/config.py` (Pydantic settings):

- `DATABASE_URL` (default SQLite in `data/app.db`)
- `STORAGE_ROOT` (default `data/`)
- `MAX_UPLOAD_MB` (default 20)
- `JWT_SECRET`, `JWT_ALGORITHM`, `ACCESS_TOKEN_EXPIRE_MINUTES`
- `SEED_ADMIN_EMAIL`, `SEED_ADMIN_PASSWORD`
- `TAVILY_API_KEY`, `RESEARCH_MAX_RESULTS`, `RESEARCH_SEARCH_DEPTH`
- `OPENAI_API_KEY`, `OPENAI_MODEL`

---

13) Suggested end-to-end milestone walkthrough

1. **Milestone 1**: signup → login → create project → upload PDF
2. **Milestone 2**: start backlog run → poll events → fetch research appendix
3. **Milestone 3**: generate epics → approve epic batch
4. **Milestone 4**: generate stories for an approved epic → approve story batch
5. **Milestone 5**: open spec WebSocket → generate spec for an approved story → approve spec

---

14) Known gaps (by assignment scope)

- No LangGraph checkpointing/pause/resume orchestration (Milestone 7/stack target)
- No pgvector semantic search
- No Langfuse observability integration
- No Milestone 6 implementation/codegen/validation pipeline

If you want, I can add a short “roadmap to Milestone 6–7” section as well.