The Haunted Reader — Kiroween Hackathon Package

This document contains the deliverables you requested for the **Haunted Reader** project: actual spec files, steering docs (Dickens, Austen, Poe, Villain, Demon), sample hook code, a complete `README.md` for the repo, ASCII UI wireframes, frontend component architecture, and backend API routes.

Table of contents

---

1) Kiro spec files

Below are the main specs to place into `/.kiro/specs/`.

1.1 `haunted_reader.yaml`

```
# /.kiro/specs/haunted_reader.yaml
name: haunted_reader
version: 1.0
description: |
  Master orchestrator spec for The Haunted Reader. Defines modules, inputs,
outputs, and expected behaviors.
modules:
  - id: upload_parser
    description: Parse uploaded files into structured chapters and plain
text.
    inputs:
      - file: [pdf, epub, txt, md, docx]
    outputs:
      - plain_text: string
      - chapters: list
      - structure: object
    behavior:
      - extract_text: true
      - segment_chapters: true
      - infer_chapter_titles: true

  - id: structural_analysis
    description: Analyze narrative structure, tension, pacing, themes and
characters.
    inputs:
      - plain_text: string
      - chapters: list
    outputs:
      - tension_map: list
      - pacing: object
      - themes: list
      - characters: list
    behavior:
      - detect_pov_shifts: true
      - detect_tension_spikes: true
      - extract_character_entities: true

  - id: summary_engine
    description: Provide multiple summary styles.
    inputs:
      - text_segment: string
      - style: [short, bullets, academic, villain]
    outputs:
      - summary: string

  - id: ghost_rewriter
    description: Rewrite or extend text in the voice of a ghost persona.
    inputs:
      - text_segment: string
      - persona_id: string
      - mode: [rewrite, continue, condense, expand]
      - constraints: object
    outputs:
      - rewritten_text: string

  - id: perspective_interpreter
    description: Produce interpretations from chosen viewpoints.
    inputs:
      - text_segment: string
      - perspective: string
    outputs:
      - interpretation: string

  - id: ending_generator
    description: Produce alternative endings.
    inputs:
      - chapter_context: string
      - ending_style: [hopeful, tragic, horror, poetic]
    outputs:
      - ending_options: list
  - id: realtime_editor
    description: Minimal collaborative editing surface with diff-aware ghost
rewrites.
    inputs:
      - original_text: string
      - selected_segment: string
      - persona_id: string
    outputs:
      - live_rewrite: string

hooks:
  - name: onUpload
    trigger: upload_parser:file_uploaded
    actions:
      - run: structural_analysis
      - run: summary_engine(style: short)

  - name: onPersonaChange
    trigger: user:persona_selected
    actions:
      - run: ghost_rewriter

  - name: onHighlightRewrite
    trigger: user:segment_selected
    actions:
      - run: realtime_editor

mcp:
  - name: text_parser
    manifest: /.kiro/mcp/text_parser/manifest.json

outputs:
  - registry: haunted_reader_outputs
```

~~~
---

## 1.2 `upload_parser.yaml`

```yaml
# /.kiro/specs/upload_parser.yaml
id: upload_parser
version: 1.0
inputs:
  - file
outputs:
  - plain_text
  - chapters
  - structure
requirements:
~~~

```
  - preserve_line_breaks: true
  - keep_metadata: [title, author, date]
steps:
  - step: validate_file_type
  - step: run_ocr_if_needed
  - step: extract_text
  - step: segment_into_chapters
  - step: infer_titles
  - step: return_json
```

~~~
---

## 1.3 `ghost_personas.yaml`

```yaml
# /.kiro/specs/ghost_personas.yaml
personas:
  - id: dickens
    name: "Charles Dickens (Ghost)"
    style: "Victorian, long descriptive sentences, moral clarity, vivid
character names"
    preferred_sentence_length: [18, 35]
    vocabulary_tone: "formal, archaic"

  - id: austen
    name: "Jane Austen (Ghost)"
    style: "social observation, irony, restrained wit, period diction"
    preferred_sentence_length: [12, 28]
    vocabulary_tone: "elegant, measured"

  - id: poe
    name: "Edgar Allan Poe (Ghost)"
    style: "gothic, recursive phrasing, dark imagery, unreliable narrator"
    preferred_sentence_length: [10, 30]
    vocabulary_tone: "melancholic, ominous"

  - id: villain
    name: "The Villain"
    style: "self-justifying, rationalizing, darkly charismatic"
    traits:
      - cynical
      - strategic

  - id: demon
    name: "The Demon"
    style: "surreal, visceral, unsettling metaphors"
    traits:
      - chaotic
      - uncanny
~~~

~~~
---

## 1.4 `summary_rules.yaml`

```yaml
# /.kiro/specs/summary_rules.yaml
rules:
  - id: short
    max_tokens: 120
    objective: "concise summary covering main plot points"

  - id: bullets
    max_items: 6
    objective: "bullet highlights with character arcs"

  - id: academic
    objective: "formal abstract highlighting themes and methodology"

  - id: villain
    objective: "inverted moral framing from villain's perspective"
~~~

~~~
---

# 2) Steering docs

Put these in `/.kiro/steering/`. Each YAML defines tone, constraints, banned
words, and patterns to follow for Kiro responses when that persona is
selected.

---

## 2.1 `base_steering.yaml`

```yaml
# /.kiro/steering/base_steering.yaml
name: base_steering
description: Default guiding rules for all personas.
constraints:
  - max_response_length: 800
  - avoid_repeating_phrases: true
  - do_not_generate_disallowed_content: true
style_rules:
  - avoid_personal_data: true
  - keep_readability: true
~~~

````
---

## 2.2 `ghost_dickens.yaml`
````

~~~
```yaml
# /.kiro/steering/ghost_dickens.yaml
persona: dickens
tone: "grand, moral, descriptive"
preferred_sentence_length: [18, 35]
dos:
  - use_period_details
  - emphasize_character_morality
  - use_vivid_character_names
donts:
  - avoid_modern_slang
  - do_not_break_period_consistency
examples:
  - input: "The boy walked into the room"
    output: "The lad, with a countenance more pale than the morning mist,
entered the chamber, and every eye was upon him."
~~~

~~~
---

## 2.3 `ghost_austen.yaml`

```yaml
# /.kiro/steering/ghost_austen.yaml
persona: austen
tone: "witty, observational, subtly ironic"
preferred_sentence_length: [12, 28]
dos:
  - prefer_social_interaction_descriptions
  - highlight_reputation_and_manners
donts:
  - avoid_explicit_gothic_tropes
examples:
  - input: "He proposed"
    output: "He offered himself, as men of sense and feeling are sometimes
invited to offer, with that mixture of earnestness and awkwardness which the
occasion invariably produces."
~~~

~~~
---

## 2.4 `ghost_poe.yaml`

```yaml
# /.kiro/steering/ghost_poe.yaml
persona: poe
tone: "gothic, claustrophobic, melodic repetition"
preferred_sentence_length: [10, 30]
dos:
  - use_repetition_for_urgency
~~~

```
  - insert_dark_imagery
donts:
  - avoid_lighthearted_jokes
examples:
  - input: "The raven perched"
    output: "The raven perched—perched upon the pallid bust—its shadow a
cipher of my ruin."
```

~~~
---

## 2.5 `ghost_villain.yaml`

```yaml
# /.kiro/steering/ghost_villain.yaml
persona: villain
tone: "confident, rationalizing, seductive"
dos:
  - justify_actions_cleverly
  - highlight_long_term_plans
  - make_motives_plausible
donts:
  - avoid_one_dimensional_evil
examples:
  - input: "I stole the map"
    output: "I acquired the map because necessity demands resources be placed
in capable hands; the world depends on those who can wield it."
~~~

~~~
---

## 2.6 `ghost_demon.yaml`

```yaml
# /.kiro/steering/ghost_demon.yaml
persona: demon
tone: "surreal, unsettling, metaphor-rich"
dos:
  - use_visceral_metaphor
  - destabilize_the_reader
  - bend_syntax_deliberately
donts:
  - avoid_hate_speech_or_excessive_gore
examples:
  - input: "He felt fear"
    output: "Fear crawled under his skin like slow ink, pooling into the
hollows that once were laughter."
~~~

```
---
```

~~~
# 3) Sample hook code

Place hooks in `/.kiro/hooks/`. Below are Node.js examples using a typical
Kiro hook environment (pseudo-API). Adjust to your real runtime.

---

## 3.1 `onUpload.js`

```javascript
// /.kiro/hooks/onUpload.js
// Triggered when a file is uploaded. Runs structural analysis and short
summary.
module.exports = async function onUpload(event, kiro) {
  // event contains { fileId, userId }
  const { fileId } = event;

  // 1) Ask MCP to parse the file
  const parsed = await kiro.mcp.invoke('text_parser', { fileId });

  // 2) Run structural analysis
  const structure = await kiro.runModule('structural_analysis', {
    plain_text: parsed.plain_text,
    chapters: parsed.chapters
  });

  // 3) Create a short summary
  const summary = await kiro.runModule('summary_engine', {
    text_segment: parsed.plain_text,
    style: 'short'
  });

  // 4) Store results for UI
  await kiro.store.set(`uploads:${fileId}:structure`, structure);
  await kiro.store.set(`uploads:${fileId}:summary`, summary);

  // 5) Notify the user/UI to refresh
  await kiro.notify(event.userId, {
    type: 'upload_ready',
    fileId,
    structureSummary: summary
  });
};
~~~

3.2 `onPersonaChange.js`

```javascript
// /.kiro/hooks/onPersonaChange.js
module.exports = async function onPersonaChange(event, kiro) {
```

```
  // event: { userId, personaId, selectedSegment }
  const { personaId, selectedSegment } = event;

  // Run ghost_rewriter
  const rewritten = await kiro.runModule('ghost_rewriter', {
    text_segment: selectedSegment,
    persona_id: personaId,
    mode: 'rewrite'
  });

  // send updated text back to client
  await kiro.notify(event.userId, {
    type: 'persona_rewrite_complete',
    personaId,
    rewritten
  });
};
```

3.3 `autoRewrite.js` (optional scheduled hook)

```
// /.kiro/hooks/autoRewrite.js
// Periodic job: apply a 'light edit' in the chosen persona for drafts that
requested auto-edit.
module.exports = async function autoRewrite(job, kiro) {
  const drafts = await kiro.store.query('drafts:needs_autorewrite');
  for (const draft of drafts) {
    const rewritten = await kiro.runModule('ghost_rewriter', {
      text_segment: draft.text,
      persona_id: draft.persona || 'dickens',
      mode: 'condense'
    });
    await kiro.store.set(`drafts:${draft.id}:autorewrite`, rewritten);
    await kiro.notify(draft.userId, { type: 'draft_autorewritten', id:
draft.id });
  }
};
```

---

4) `README.md`

Place this in project root as `README.md`.

```
# The Haunted Reader

**The Haunted Reader** — an AI-powered literary engine that summons ghostly
```

```
perspectives to analyse, reinterpret, and finish texts.

## Live demo
Insert your deployed URL here.

## Categories
Primary: **Frankenstein** — merges AI, parsing, UX, and style-transfer
modules.

## Tech stack
- Frontend: React + Vite (or Next.js) + Tailwind
- Backend: Node.js (Express) or Fastify
- AI: OpenAI / local LLM via API
- Kiro integration: /.kiro specs, hooks, steering docs, MCP
- Storage: PostgreSQL (metadata), S3 (uploads)

## Repo structure
```

/src /client /server /README.md /.kiro /specs /hooks /steering /mcp

```
## How to run (dev)
1. Install dependencies: `pnpm install` or `npm install`
2. Start server: `pnpm dev:server`
3. Start client: `pnpm dev:client`
4. Visit `http://localhost:3000`

## Kiro integration
- `/.kiro/specs` contains specs for upload parsing, analysis, persona
rewrites, and summaries.
- `/.kiro/hooks` contains automation hooks that respond to uploads and
persona changes.
- `/.kiro/steering` contains the steering docs for each persona.
- `/.kiro/mcp` contains the manifest and parser implementation for uploads.

## Demo checklist for judges
- Upload a PDF/EPUB → see structure and short summary
- Select a passage → apply Dickens persona → see rewrite
- Try multiple personas on the same passage to compare
- Use Generate Ending to create 3 endings

## License
Choose an OSI-approved license (MIT recommended).

## Contributing
PRs welcome. See CONTRIBUTING.md for style and commit rules.
```

```
---

# 5) ASCII UI wireframes
```

```
Use these for a first-pass implementation and to guide CSS/HTML layout.

---

## 5.1 Main app layout (wide desktop)
```

+---+ | HEADER: The Haunted Reader [Upload] [Persona carousel] [Help] | +---+ | LEFT: DOCUMENT (70%) | RIGHT: GHOST PANEL (30%) | | --|-----------------------------------| | Title: [Chapter 3] | [Ghost Carousel] | | [Parchment background] | (circle) Dickens (circle) Poe | | | [Interpretation Title] | | 1| Lorem ipsum dolor sit amet... | -> [summary / rewrite panel] | | 2| More text... | [Buttons: Rewrite / Continue] | | 3| Selected passage highlighted | [Tone sliders: archaic / modern] | | | [Generate ending] | +---+ | FOOTER: Séance Bar [Interpret] [Possess] [History] [Settings] | +---+

```
---

## 5.2 Mobile (stacked)
```

+--------------------------+ | HEADER: Haunted Reader | +--------------------------+ | [Upload] | +--------------------------+ | Document (scroll) | | [select passage] | +--------------------------+ | Persona carousel (scroll) | +--------------------------+ | Interpretation (accordion)| +--------------------------+

```
---

# 6) Frontend component architecture

A React component map and responsibilities. Use TypeScript.
```

/src/client /components - AppShell.tsx // layout, header, footer - UploadDropzone.tsx // file uploads, progress - DocumentViewer.tsx // shows parsed text, allows selection - ChapterList.tsx // list of chapters, jump-to - PersonaCarousel.tsx // choose ghost persona - InterpretationPanel.tsx// shows summaries, rewrites, endings - EditorPane.tsx // realtime editor with diff - TensionGraph.tsx // visualizes tension map - SeanceBar.tsx // bottom controls - Modal/Confirm.tsx /hooks - useUpload.ts - usePersonas.ts - useRealtimeRewrite.ts /styles - tailwind config /pages (if Next.js)

```
**Data flow**
- UploadDropzone -> calls API `/api/upload` -> returns parsed chapters
- DocumentViewer shows text, selection -> triggers `/api/interpret` or
triggers Kiro hook via websocket
- PersonaCarousel selects persona -> UI calls `/api/ghost_rewrite` or uses
Kiro agent via client

**State management**
- Use React Query for server state
- Use local state for selection/highlights
- Use WebSocket for live rewrite notifications
```

```
---

# 7) Backend API routes

Design these Express-style routes. Implement with Node/Express or Fastify.
```

POST /api/upload - body: multipart file - action: store file -> call MCP parser -> return { fileId, chapters, summary }

GET /api/uploads/:fileId/structure - returns stored structure (chapters, tension_map)

POST /api/interpret - body: { fileId, segment, perspective } - action: run perspective_interpreter -> returns interpretation

POST /api/ghost_rewrite - body: { fileId?, segment, persona_id, mode } - action: run ghost_rewriter -> returns rewritten_text

POST /api/generate_ending - body: { fileId?, chapter_context, style } - action: run ending_generator -> returns list of endings

WS /ws - purpose: subscribe to notifications (upload_ready, persona_rewrite_complete)

GET /api/personas - returns list of available persona metadata and steering IDs

POST /api/save_draft - saves edited text to DB

POST /api/request_autorewrite - schedules the draft for autoRewrite job ```

Auth - JWT-based session with refresh tokens - Optional social login (GitHub/Google)

Storage - Files -> S3-compatible bucket - Metadata -> PostgreSQL - Rewrites/history -> Postgres or Redis for fast access

---

Final notes & next steps

- All specs in `/.kiro/specs` should be treated as the canonical source of truth for Kiro to build against.
- Steering documents must be iterated after you test outputs to tune persona voice.
- The MCP `text_parser` should handle OCR for PDFs (Tesseract or cloud OCR) and EPUB parsing (epub.js or python ebooklib).
- Start with a minimal viable persona set: `dickens`, `poe`, `villain`, `demon`, `neutral` and expand later.
- For the hackathon, focus the demo flow: upload -> analysis -> persona rewrite -> generate ending.

If you'd like, I can now: - generate actual files in a repo structure (Y/N) - produce the MCP parser implementation in Python - produce Figma-ready design tokens (colors, fonts, spacing)

Tell me which to generate next and I will create the files.