"AI-powered Content Extractor"

Project Summary

Create a full-stack web app that:

- Allows users to input any **public URL**
- Extracts meaningful **text content** from that webpage
- Uses **open-source NLP techniques or simple summarization logic** (no paid APIs like OpenAI)
- Generates a **summary and list of key points**
- Displays the results in a **Notion-like table view** (searchable, sortable, filterable)
- Is ready for **deployment on Vercel (frontend)** and **runs Java + Spring Boot backend**
- Codebase must be **GitHub-ready with documentation**

Tech Stack Constraints

Use **only** the below technologies unless absolutely necessary:

- **Backend**: Java, Spring Boot, REST APIs, Hibernate
- **Frontend**: React.js, Tailwind CSS or CSS3, shadcn/ui (for Notion-style tables)
- **HTML Parser**: JSoup or Apache Tika (Java)
- **AI Summary**: Use basic NLP (e.g., TF-IDF, frequency-based, or open-source summarizer) – no paid APIs
- **Deployment**: Vercel for frontend, backend ready for direct run
- **Dev Tools**: Git, Maven
- **Security**: Apply good practices (CORS, input validation, REST standards)

Functional Requirements

Core Features:

1. **Frontend (React.js)**:
    - Text input box for public URL
    - Submit triggers API request
    - Show:
        - Clean **summary**
        - List of **key points** in a table (Notion-style)
    - Table should allow:
        - Search, filter, edit, delete
        - Pagination (optional)
        - Export to .csv or .json (optional)
2. **Backend (Spring Boot)**:
    - Endpoint: POST /api/extract

        - Input: { url: string }
        - Validates URL and fetches HTML
        - Uses JSoup or Apache Tika to extract main content (removes nav/footer/ads)
        - Applies **free summarization logic**
            - E.g., frequency-based scoring of sentences
            - Extract top N sentences + bullets
        - Responds with:
          {
          summary: "string",
          keyPoints: ["point 1", "point 2", ...]
          }
    - Handles edge cases:
        - Invalid URLs, timeouts, CORS issues
        - Empty pages, JavaScript-heavy pages (return informative error)
        - Multiple redirects

Architecture

```
frontend/
  └── React app (Next.js structure)
      └── components/
          └── UrlInput.jsx
          └── SummaryBox.jsx
          └── KeyPointsTable.jsx
      └── pages/
          └── index.jsx
backend/
  └── Java Spring Boot app
      └── controllers/
          └── ExtractController.java
      └── services/
          └── HtmlExtractorService.java
          └── SummarizationService.java
      └── utils/
          └── JsoupUtils.java
      └── models/
          └── SummaryResponse.java
```

Testing Strategy

- Backend:
    - JUnit tests for SummarizationService
    - Integration test for /api/extract (mock URL input)
- Frontend:

    - Component-level tests using @testing-library/react
    - Test: URL input submission, summary rendering, table filtering
- Manual test checklist in README

Security

- Input sanitization
- URL validation (must be HTTP/S)
- CORS enabled only for frontend origin
- Exception handler for backend errors (return 4xx/5xx with messages)

Documentation: README.md

Include:

1. Project Overview
2. Technologies used
3. How to run locally
4. How to deploy frontend to Vercel
5. Backend API description with sample request/response
6. Running tests
7. Folder structure
8. Ideas for future enhancements (e.g., Auth, History, Exporting, Cloud AI)

Follow-Up Tasks

1. **Write full README.md**
2. Add inline **comments** for:
    - HtmlExtractorService
    - SummarizationService
    - KeyPointsTable component
3. **Create a vercel.json** file if required for proxying API to backend
4. Output a GitHub-ready project (use git init, add .gitignore, structure cleanly)
5. Write 1-2 JUnit tests for backend
6. Provide a mocked sample run with a real public article URL (e.g., Wikipedia)

Deployment Instructions (Add to README too)

Frontend:

- Push React app to GitHub
- Connect repo to Vercel
- Set environment variable for backend API URL (e.g., NEXT_PUBLIC_API_BASE_URL)

Backend:

- Java Spring Boot app runs on port 8080
- Can be run locally (mvn spring-boot:run)