List of Tasks for the Alai Challenge

1. Project Setup

* Task 1.1: Install Node.js (if not already installed) and verify the version (node -v).
* Task 1.2: Create a new project directory (alai-challenge) and initialize it with npm init -y.
* Task 1.3: Install core dependencies:
  + Run npm install @mendable/firecrawl-js axios cheerio jsonwebtoken dotenv.
* Task 1.4: Create the folder structure as outlined earlier:
  + src/api/, src/auth/, src/utils/, src/config/, src/index.js, tests/, .env, README.md.
* Task 1.5: Set up .env file:
  + Add FIRECRAWL\_API\_KEY=your-key-here (placeholder for now).
* Task 1.6: Configure package.json scripts:
  + Add "start": "node src/index.js" and "test": "jest" (optional, if adding tests).
* Task 1.7: Install a testing framework (optional):
  + Run npm install --save-dev jest and set up tests/ directory.

2. Account and API Preparation

* Task 2.1: Sign up for a Firecrawl account at [firecrawl.dev](https://www.firecrawl.dev/).
* Task 2.2: Obtain a free Firecrawl API key from the dashboard and update .env with FIRECRAWL\_API\_KEY.
* Task 2.3: Sign up for an Alai account at [www.getalai.com](https://www.getalai.com).
* Task 2.4: Manually create a sample presentation on Alai:
  + Use Chrome DevTools (Network tab) to monitor API requests while creating a 2-5 slide presentation.
* Task 2.5: Identify Alai API endpoints:
  + Note endpoints for authentication (token), slide creation, and presentation generation (e.g., /auth, /slides, /publish).
  + Record request headers (e.g., Authorization: Bearer <token>), payloads, and responses.
* Task 2.6: Save the Alai API base URL (e.g., https://app.getalai.com/api) in src/config/config.js.

3. Implement Web Scraping

* Task 3.1: Write src/api/firecrawl.js:
  + Initialize Firecrawl SDK with the API key from .env.
  + Create a function scrapeUrl(url) to fetch markdown or structured data from a webpage.
* Task 3.2: Test Firecrawl integration:
  + Call scrapeUrl("https://example.com") and log the output to verify it works.
* Task 3.3: Write src/utils/scraper.js:
  + Create a function processScrapedData(data) to clean or format Firecrawl output for slides (e.g., split markdown into sections).

4. Implement Authentication for Alai

* Task 4.1: Write src/auth/tokenManager.js:
  + Create a TokenManager class with methods getToken() and refreshToken().
  + Use axios to fetch a token from the Alai auth endpoint (identified in Task 2.5).
  + Store token and expiration time (assume 30 minutes initially).
* Task 4.2: Add token refresh logic:
  + Check expiration before each API call and refresh if needed (use jsonwebtoken to decode/verify if applicable).
* Task 4.3: Test token fetching:
  + Log the token and ensure it works with a manual Alai API call (e.g., via Postman or curl).

5. Implement Alai Presentation Creation

* Task 5.1: Write src/api/alai.js:
  + Create functions like createPresentation(token, slides) and getShareableLink(presentationId) based on endpoints from Task 2.5.
  + Use axios to chain API calls: create a draft, add slides, publish presentation.
* Task 5.2: Format slide data:
  + Convert scraped content (from scraper.js) into a 2-5 slide structure (e.g., { title: "Slide 1", content: "text" }).
* Task 5.3: Test Alai integration:
  + Manually pass sample slide data and verify a shareable link is returned (e.g., https://app.getalai.com/view/...).

6. Tie It All Together

* Task 6.1: Write src/index.js:
  + Accept a URL as input (e.g., via command-line argument or hardcoded for testing).
  + Call scraper.js to scrape the webpage, tokenManager.js for authentication, and alai.js to create the presentation.
  + Output the shareable link to the console.
* Task 6.2: Add basic error handling:
  + Wrap API calls in try-catch blocks and log meaningful errors (e.g., “Failed to scrape URL”).
* Task 6.3: Test end-to-end:
  + Run npm start with a sample URL (e.g., node src/index.js https://example.com) and verify the output link works.

7. Extra Credit Enhancements

* Task 7.1: Write src/utils/parser.js:
  + Use cheerio to parse Firecrawl’s HTML output and extract images or metadata.
* Task 7.2: Enhance slide content:
  + Add logic in scraper.js to split content into better-organized slides (e.g., by headings or paragraphs).
* Task 7.3: Import images into Alai:
  + Check if Alai’s API supports image uploads (via Network tab).
  + If yes, download images with axios and include them in slide payloads in alai.js.
* Task 7.4: Test enhancements:
  + Run the script with a webpage containing images and verify they appear in the presentation.

8. Testing and Validation

* Task 8.1: Write basic tests in tests/api.test.js:
  + Test firecrawl.js and alai.js functions with mock data or real API calls (if rate limits allow).
* Task 8.2: Write tests in tests/auth.test.js:
  + Test tokenManager.js for token fetching and refresh scenarios.
* Task 8.3: Run tests:
  + Use npm test to ensure all components work independently.

9. Documentation and Submission

* Task 9.1: Update README.md:
  + Add setup instructions (e.g., npm install, .env setup), usage (npm start <url>), and a project overview.
* Task 9.2: Record a 1-2 minute Loom video:
  + Explain your approach: setup, Firecrawl scraping, Alai API reverse-engineering, token management, and enhancements.
  + Demo the script with a sample URL and show the resulting presentation.
* Task 9.3: Final validation:
  + Test the script with multiple URLs to ensure robustness.
  + Open the generated presentation links to confirm they work as expected.
* Task 9.4: Submit the project:
  + Package the code (zip the folder) and include the Loom video link per challenge instructions.

Estimated Timeline

* Setup (Tasks 1-2): 1-2 hours
* Web Scraping (Task 3): 1 hour
* Authentication (Task 4): 1-2 hours
* Presentation Creation (Task 5): 2-3 hours
* Integration (Task 6): 1 hour
* Extra Credit (Task 7): 1-2 hours (optional)
* Testing (Task 8): 1 hour (optional)
* Documentation (Task 9): 1-2 hours
* Total: ~8-14 hours, depending on familiarity and enhancements.

Notes

* Reverse-Engineering Alai: Tasks 2.4-2.5 are critical and may take longer if the API is complex. Save all network requests as a HAR file for reference.
* Error Handling: Add as you go to avoid debugging headaches later.
* Iterate: Test each module (e.g., scraping, token fetch) before integrating to catch issues early.

This list breaks the challenge into manageable steps, ensuring you cover every requirement from setup to submission. Let me know if you need help with any specific task!

