Backend Enhancement Assessment

Project Summary

This project is a Node.js + JavaScript backend built with Express that provides AI-powered APIs and basic user management functionality. The system supports AI endpoints (chat, text generation, sentiment analysis, summarization) and user CRUD operations with roles (user/admin). It includes core utilities such as caching, rate limiting, input validation (Zod), error handling, and structured logging. It works with or without an OpenAI API key (using mock AI responses if no key is provided). The backend can run locally or within Docker environments and works instantly with the mock AI setup.

Assessment Objective

Enhance the backend's observability, flexibility, and monitoring capabilities by implementing three incremental features. Each task builds upon the existing architecture patterns and follows the JavaScript + Express.js conventions already in place.

Task 1: Add Cache Statistics Endpoint

- Create a new route GET /api/cache/stats that returns cache information.
- Use the existing cacheService.size() method (already available).
- Add a getStats() method to CacheService that returns useful statistics such as cache size, default TTL, and optionally hit/miss tracking.
- Create a cache controller to handle the endpoint.

Files to Modify: src/routes/index.js, src/services/cache.service.js, src/controllers/cache.controller.js, src/routes/cache.routes.js

Example Response: Returns JSON with cache size, max size, default TTL, and hit/miss statistics if implemented.

Task 2: Add Model Selection to AI Endpoints

- Update the chat and generate endpoints to accept and use an optional model parameter.
- Validate the model selection against available models defined in AI_MODELS (GPT_3_5_TURBO, GPT_4, GPT_4_TURBO).
- Update aiService.chatCompletion() and aiService.generateText() to accept and use the model parameter.
- Modify OpenAIService.chatCompletion() to dynamically use the provided model (currently hardcoded to 'gpt-3.5-turbo').

Files to Modify: src/controllers/ai.controller.js, src/services/ai.service.js

Example Request: Includes an optional 'model' field validated against AI_MODELS before execution.

Task 3: Add Request ID Tracking Middleware

- Create middleware that generates a unique ID for each request using uuid or crypto.randomUUID().
- Attach the request ID to req.requestId and response headers (X-Request-ID).
- Enhance the requestLogger middleware and logger utility to include the request ID in log messages.
- Integrate the middleware in src/index.js before other middleware to ensure global coverage.

Files to Modify: src/middleware/requestLogger.js, src/index.js, src/utils/logger.js

Example: Logs include [abc123] Request received or [INFO] [abc123] GET /api/ai/chat 200 - 150ms

Key Implementation Notes

- This is a JavaScript project; all files use .js (not .ts).
- CacheService already includes a size() method — only extend functionality for stats if needed.
- The AI controller includes a 'model' field in chatSchema that must now be validated and passed through.
- The OpenAIService.chatCompletion() currently hardcodes 'gpt-3.5-turbo' and must be made dynamic.
- Follow the established architecture patterns: separate controllers, routes, and singleton services.