```
Staff Interview - Possible Exercises and Instructions
Generated: Thu Feb 19 09:36:30 CST 2026

============================================================
1) QUICK RUN INSTRUCTIONS (PROJECT READY)
============================================================

Prerequisites
- Node.js 18+
- npm

Run backend
1. cd /Users/fmoya/Documents/Staff Interview/backend
2. npm install
3. npm run dev
4. Verify: http://localhost:4000/health

Run frontend
1. cd /Users/fmoya/Documents/Staff Interview/frontend
2. npm install
3. npm run dev
4. Open: http://localhost:3000/posts

Environment
- frontend/.env.local
  NEXT_PUBLIC_API_BASE=http://localhost:4000

Quick checks before interview
- Backend responds on /health and /api/posts
- Frontend loads /posts and /posts/:id
- Create/Edit flow works from UI
- If backend is down, frontend shows graceful error
- IDE ready, screen share ready, internet stable

============================================================
2) INTERVIEW EXERCISES (LIKELY)
============================================================

Exercise 1: Add endpoint GET /users/:id/posts
Goal
- Extend backend routes with validation and error handling.
Expected
- 200 with { posts }, 400 invalid id, 500 upstream failure.

Exercise 2: Add pagination to GET /posts
Goal
- Support page and limit query params.
Expected
- Validate query params and return metadata { page, limit, total }.

Exercise 3: Add search filter by title
Goal
- /posts?search=keyword
Expected
- Case-insensitive matching and consistent response shape.

Exercise 4: Implement DELETE /posts/:id
Goal
- Add route + status code conventions.
Expected
- 204 on success, 400 invalid id, 500 on errors.
Note
```

- JSONPlaceholder does not persist deletions.

Exercise 5: Explain and demonstrate N+1
Goal
- Show bad pattern vs optimized pattern.
Expected
- n+1: posts + comments per post.
- optimized: batched comments query and group by postId.

Exercise 6: Add PUT /posts/:id (full update)
Goal
- Explain PUT vs PATCH semantics.
Expected
- PUT validates full payload; PATCH validates partial payload.

Exercise 7: Improve frontend error/empty/loading states
Goal
- Better UX resilience.
Expected
- Friendly error blocks, empty state message, loading indicator.

Exercise 8: Add request logging middleware
Goal
- Improve observability.
Expected
- Log method, path, status, and duration.

Exercise 9: Refactor backend to Controller-Service structure
Goal
- Improve maintainability.
Expected
- Routes call controllers; logic moved to service layer.

Exercise 10: Add simple auth guard concept (mock JWT)
Goal
- Show auth/authz knowledge.
Expected
- Check Bearer token, return 401/403 correctly.

==
3) EXAMPLE ANSWER FRAMEWORK (HOW TO EXPLAIN)
==

When solving each exercise, explain in this order:
1) Requirement understanding
2) API contract (input/output/status codes)
3) Validation and edge cases
4) Implementation plan
5) Tradeoffs/performance
6) Testing approach

Example short script
"I'll validate inputs first, keep response contracts consistent,
use correct HTTP status codes, and add a minimal test path for
success and error cases. Then I'll explain tradeoffs and next steps."

==
4) COMMON MISTAKES TO AVOID
==

- Start coding without clarifying API contract.
- No input validation.

- Wrong status codes (e.g., always 200).
- Ignoring error handling and retries.
- Mixing concerns (route logic + business logic + API client in one place).
- Not testing the endpoint manually before saying it is done.

5) FINAL 15-MIN CHECKLIST (BEFORE INTERVIEW)

- Backend running on :4000
- Frontend running on :3000
- /api/posts returns data
- /posts loads in UI
- One create/edit flow demonstrated
- Terminal and editor prepared
- Postman/curl command ready