

Technical Exercise

Graduate Software Developer, AI & Automation – Technical Exercise

From: info@thirdwavebbq.com.au | Sat, Sep 5, 2026 at 7:34 PM

To: leonardo.prasetyo5@gmail.com

Hi Leonardo,

Thank you for applying for the Graduate Software Developer, AI & Automation position with Third Wave BBQ.

We are looking for someone who will use AI coding tools as a primary development method, while remaining responsible for understanding the system, reviewing the generated code, testing it, deploying it and resolving problems.

This exercise is designed to understand how you approach software development using AI. You are encouraged to use Codex, Claude Code, Cursor, GitHub Copilot, ChatGPT or any other AI tools.

Please spend no more than 45 minutes on the exercise. We are not asking you to build the feature. Please keep your response to a maximum of approximately 1,200 words.

Scenario

Third Wave BBQ has an existing web application used by Head Office and multiple restaurant locations.

The application uses:

- Next.js for the frontend
- NestJS for the backend
- PostgreSQL for the database
- Docker Compose
- An Ubuntu VPS

We want to add a maintenance and operational issues feature.

A venue manager should be able to:

- Create an issue
- Add a description and photographs
- Select a priority
- Assign the issue to another user
- Set a due date
- Update its status
- Add comments

Venue employees must only be able to see issues belonging to their own venue. Head Office employees must be able to see issues across all venues.

Overdue issues should generate a daily email reminder.

Please respond to the following

1. Requirements

List up to eight questions you would ask before starting development. Where information is missing, state the assumptions you would make so that you could continue working.

2. Development plan

Explain how you would break the feature into manageable stages. Briefly cover database structure, backend and API, frontend, user permissions, photograph storage, email reminders, testing and deployment. We are more interested in your reasoning than in detailed code.

3. Instructions to the AI coding agent

Provide the exact first instruction or prompt you would give an AI coding agent after opening the existing repository. Your instruction should tell the agent what to inspect, what to build, what constraints to follow and what must be true before the work is considered complete. Also provide one follow-up instruction you might give after reviewing the agent's first attempt.

4. Reviewing AI-generated work

Explain how you would check the changes before accepting them. Include what files and code changes you would inspect, what automated checks you would run, what you would test manually, what security and permission issues you would check, and how you would identify unnecessary or suspicious dependencies. Include specific commands or tools where you know them.

5. Code review

Consider the following simplified endpoint:

```
app.get('/api/issues', requireLogin, async (req, res) => {
  const venueId = req.query.venueId;
  const issues = await db.issue.findMany({
    where: venueId ? { venueId } : {}
  });
  res.json(issues);
});
```

Explain what could be wrong or unsafe about this approach, which issue is the most serious, how you would correct it, and how you would test that the correction works. You do not need to rewrite the complete endpoint.

6. Deployment

Explain how you would deploy the feature to the VPS without unnecessarily risking the existing production application or database. Cover Git branches and code review, database backups, database migrations, environment variables and secrets, Docker deployment, application logs, health checks, and rollback if the release fails.

7. Production problem

After deployment, a venue manager reports that they can see an issue belonging to another restaurant. At approximately the same time, some users begin receiving HTTP 500 errors.

Explain what you would do immediately, which problem you would prioritise, what information and logs you would inspect, how you would reproduce the problem, how you would determine whether the issue was caused by the application, database, permissions or deployment, and what you would do before releasing a correction.

8. Previous project

Provide an example of one software project or automation you have built using AI. Explain what the project did, which parts you were personally responsible for, which AI tools you used, how it was hosted or deployed, something the AI initially got wrong, and how you identified and corrected the problem. Provide a link to the project or GitHub repository where available.

AI usage disclosure

Please include the AI tool or tools you used for this exercise, the relevant prompts you gave them, any important AI recommendations you rejected or changed, and a brief explanation of why you made those changes.

We are not assessing whether you can complete the exercise without AI. We are assessing whether you can use AI effectively while remaining in control of the technical result.

Please return your response by 6 September 2026.

Kind regards,

Greg

Third Wave BBQ

Technical Exercise Response

Graduate Software Developer AI and Automation

Leonardo Prasetyo | Third Wave BBQ | 5 September 2026

1. Requirements and assumptions

Question	Working assumption
Should issue activity be recorded?	To confirm; record creation, assignment, resolution and commissioning.
Should descriptions support Markdown in a Notion-like editor?	To confirm; this affects image storage and uploads.
Do Maintenance and Operational issues use different workflows?	To confirm; assume one workflow initially.
Can users belong to multiple venues?	Yes.
Should new issue types be configurable?	Yes; store them as records, not code.
What can each role do?	Employees: assigned issues only. Managers: multiple restaurants. Head Office: unrestricted.
Can comments change, and is an audit trail required?	Text only; users cannot edit; audit key changes.
What scale, uptime and existing services apply?	Reuse services; no planned downtime; hundreds of open issues.

2. Development plan

Assuming I am the only developer, I would:

1. **Research established issue trackers such as GitHub Issues, adapting repository scope to restaurants.**
2. **Sketch the issue list, form, comments, photos, dates and status flow.**
3. **Review wireframes with Greg and venue stakeholders.**
4. **Design PostgreSQL tables and NestJS routes, including permissions, private photos and reminders.**
5. **Develop in short AI-assisted loops; document scope, migrations and deployment risks.**
6. **Test role isolation, uploads and reminders in staging; deploy and monitor.**
7. **Improve performance, refactor and maintain.**

Frontend Inspiration

[see attached GitHub-frontend-inspiration.pdf]

Proposed backend endpoints

Method	Route
GET · POST	/api/issues
GET · PATCH · DELETE	/api/issues/:id
GET · POST	/api/issues/tags

Server-enforce restaurantId and authorId (or the existing venueId). Register /tags before /:id.

Proposed database structure

The existing users and venues tables remain authoritative:

Table	Key columns	Purpose and constraints
issues	id uuid PK; venue_id FK; author_id FK; assignee_id FK nullable; title varchar(160); description text; priority; status; due_at timestamptz nullable; closed_at; deleted_at; created_at; updated_at	Core issue record. venue_id provides restaurant scope; deleted_at preserves history when the DELETE route is used.

The first release covers issue creation, descriptions, private photos, priority, assignment, due dates, status and comments. Employees see assigned issues; managers see their restaurants; Head Office sees all venues. The API enforces scope, private storage protects photos, and a daily job emails overdue reminders. I would test in staging and deploy with Docker Compose.

3. Instructions to the AI coding agent

First instruction

Review the repository without changes. Explain its purpose, architecture, applications and services. Cite files identifying languages, frameworks, database, infrastructure, build and test tools. Summarise features. Group pages and routes with their UI components, API handlers, services and data access. Find every AGENTS.md; state its scope, constraints and validation commands. Note documentation gaps. Return a file-linked guide; do not modify code.

Possible follow up

Using [issue-implementation-document], review the venue issue proposal against the guide and AGENTS.md. Identify reusable routes, components, services, data and permission patterns; flag conflicts, missing requirements and risks. Recommend changes to scope, API, schema, access control, tests and rollout. Do not implement. Return a revised plan and unresolved decisions.

4. Reviewing AI generated work

I would run the implementation on my local machine using the repository's Node.js processes and Docker Compose services. Then, test each page and workflow in the browser, looking for visual bugs, confusing interactions, inconsistent spacing, poor formatting and missing loading, empty or error states. I would also verify the role-based interface so each user sees only the information and actions they need. AI-generated frontends often expose too much information or produce awkward layouts, so I would simplify and polish the interface before accepting it.

Behind the interface, I would review generated React component names and boundaries, component reuse, state management and fetch lifecycles, API contracts, database tables and migrations, validation, authentication and authorisation, secret handling, logging and error handling. I would confirm that the code follows existing patterns, does not expose secrets in source, logs or client bundles, and does not add unnecessary dependencies.

5. Code review

Authentication alone does not authorise data scope. A venue employee can request another venue's ID, while omitting the parameter returns every issue. This cross-venue disclosure is the most serious defect.

I would derive scope from role and venue membership. Venue users remain limited to their venue; Head Office may request one or all venues. I would add pagination and an explicit response shape. Tests cover 401, omitted venueId, cross-venue and Head Office access, and guessed IDs across list, detail, update, comment and photo routes.

6. Deployment

First, I would run the application locally using its Node.js processes and Docker Compose services. Next, changes on the dev branch deploy automatically to a hosted development environment for cloud-hosted testing, while main represents production. Each feature/* and fix/* branch is reviewed and merged only into dev. Once the hosted environment passes functional, permission and deployment checks, a reviewed pull request from dev to main starts the production release.

The dev branch remains the canonical source for production releases. The pipeline promotes the same Docker image and compatible configuration tested in the hosted development environment, avoiding differences between the tested and production artifacts. Before promotion, the release checklist verifies a pg_dump backup, additive and backward-compatible migrations, environment variables, secrets and rollback steps. After deployment, monitoring covers migration output, Node.js, container and proxy logs, health checks and permission smoke tests. If the release fails, the previous image is redeployed; database restore remains the last resort.

7. Production problem

If an incident starts after a recent deployment and the previous release is known to be safe, the first response is an immediate rollback rather than a live production repair. This restores service quickly and reduces the risk of an untested hotfix. Before the rollback, I would preserve the deployment SHA, request IDs and relevant logs, then confirm that any database migration remains backward-compatible with the previous application image. If a rollback is unsafe, the affected feature or endpoint is disabled while the investigation continues.

For a manager seeing issues from another venue, the investigation begins with the request and API response in Chrome DevTools. If the response is correct, the next step is to trace the frontend fetch lifecycle, cache keys and store isolation for stale or cross-venue state. However, if the response already contains another venue's data, the problem is treated as a backend authorisation and query-scoping defect, and the endpoint is restricted immediately. For 500 errors, the request is traced from the browser through the application, container, proxy and database logs, then compared with the most recent deployment. Checks cover build or runtime errors, failed migrations, missing columns, configuration differences and unhealthy services. Finally, the issue is reproduced and fixed locally or in the hosted development environment, supported by regression tests, and redeployed through the normal dev-to-production pipeline.

8. Previous project

I built an AI portfolio that answers questions about my experience from published PDFs. I implemented Nuxt and TypeScript, Workers AI and AI Gateway, D1, private R2, RAG, GitHub-admin controls and durable indexing. Codex assisted while I owned architecture, review, testing and deployment.

AI first ran extraction and embedding inside the HTTP request. Local tests exposed timeout and partial-publication risks. I moved it to Cloudflare Workflows, added progress and retries, isolated vector generations, and activated documents only when vectors were queryable.

Project: leonardoprasetyo.com

Github: github.com/Beanmommel/leonardoprasetyo

Written Follow-Up Exercise

Graduate Software Developer, AI & Automation – Written Follow-Up Exercise (15–20 Minutes)

From: info@thirdwavebbq.com.au | Sun, Sep 6, 2026 at 12:35 AM

To: leonardo.prasetyo5@gmail.com

Hi Leonardo,

Thank you for taking the time to complete and send through your initial technical exercise. We appreciate the thought and effort you put into your response.

We're pleased to let you know that you have progressed to the next stage of the assessment.

Please complete the short written follow-up below and reply to this email with your answers.

Allow 15–20 minutes in total, including reading. Aim for no more than 500 words, excluding code and tests. You do not need to build an application or set up a development environment.

We are interested in how you understand a problem affecting real people, reason about code, challenge AI-generated suggestions and adapt to new information—not how polished your writing is.

THE SCENARIO

Staff use an application to report operational problems between shifts. Reports have fallen by 40%, but recurring problems have not decreased. Management wants AI-generated reminders encouraging staff to submit more reports.

1. UNDERSTAND THE PROBLEM — about 3 minutes

Before agreeing to build reminders:

- What are the two most important questions you would ask, and who would you ask?
- What evidence would help you distinguish a software defect from a workflow or trust problem?

2. REVIEW THE CODE — about 8 minutes

You now learn that staff say problems disappear from their list when the next shift acknowledges them—even when nothing has been fixed. Managers interpret the falling report count as an improvement.

An AI coding assistant proposes the following implementation and passing test. For this exercise, assume authentication and permissions are already correctly handled, and `openIssue` is an unresolved issue.

```
function acknowledge(issue, staffId) {
  issue.acknowledgedBy = staffId;
  issue.status = "resolved";
  return issue;
}
function visibleIssues(issues) {
  return issues.filter(issue => issue.status !== "resolved");
}
// AI-generated test:
expect(acknowledge(openIssue, "staff-7").status)
  .toBe("resolved");
```

- Explain what the code actually does and whether the passing test demonstrates that it meets staff needs.
- Show the smallest useful correction, using code or pseudocode. State any assumption about how an issue should become resolved.
- Give one regression test with its setup, action and expected result. Explain why it fails against the original code and passes against your correction.

3. DIRECT AND VERIFY AI — about 5 minutes

Write the exact prompt you would give an AI coding assistant to make your correction in an existing codebase. Include what it should inspect, the boundaries of the change and how it should demonstrate correctness.

Name one plausible but unsuitable AI suggestion you would reject, explain why, and describe one verification step you would perform independently rather than relying on the AI's assurance.

AI is allowed but not required. If you use it, briefly name the tool and describe what it contributed and what you checked or changed. If you do not use AI, say so and provide your proposed prompt. A paid subscription is not required.

4. ADAPT TO A NEW CONSTRAINT — about 4 minutes

Staff sometimes work offline on a shared device. An acknowledgement made while offline could arrive at the server after someone else has resolved or reopened the issue. Management still wants one simple dashboard number.

- Describe one concrete sequence where a naive implementation produces the wrong outcome, and the rule you would use to prevent it. No full offline system implementation is required.
- Choose one measure of whether your intervention helps staff. Explain one way that number could be misleading and what you would check alongside it.

YOUR REPLY

Please label your answers 1–4 and include your approximate time spent and AI-use note. Keep your response focused on decisions and reasons; a short, thoughtful answer is more useful than a lengthy specification.

If you need an adjustment to the exercise or more time to participate, please let us know.

Thank you again for your interest in joining Third Wave BBQ. We look forward to reading your response.

Kind regards,

Greg

Third Wave BBQ

Written Follow Up Exercise

Graduate Software Developer AI and Automation

Leonardo Prasetyo | Third Wave BBQ | 6 September 2026

1 Understand the problem

- To management: By AI-generated reminder do you mean an automated reminder to submit “draft” issues?
- To staff: After submitting a problem, what happens next, and is prior issues reported have been handled appropriately? (i.e. issue status progress, transparency, communication issues)

This may be a workflow/ trust problem as employees may have reported problem in between shifts, causing miscommunication between employees. Furthermore, evidence stated that the rate of report that is declining by 40% yet operational issues still occur.

2 Review the code

`acknowledge()` function records `acknowledgedBy`, but sets status to `resolved`, and causes `visibleIssues()` to hide the issue. The passing test confirms this behaviour, but not that staff needs were met.

I assume `acknowledged` is a supported status and that acknowledgement and resolution are separate actions. The smallest correction is to set the status to `acknowledged`; a separate resolution action should require a non-empty comment describing the fix.

```
function acknowledge(issue, staffId) {  
  issue.acknowledgedBy = staffId;  
  issue.status = "acknowledged";  
  return issue;  
}
```

Regression test

Setup: create an open issue. Action: acknowledge it. Expected: status is `acknowledged` and `visibleIssues` still contains it. This fails originally because the issue becomes `resolved` and disappears but passes after the correction. (on `spec.ts`)

3 Direct and verify AI

Exact prompt

Following the issue acknowledgement feature deployment, it seems to cause the issue to be “resolved”, as a result it hides acknowledge issues from the outstanding issue list. Could you inspect the issue acknowledgement workflow, identify function issue, and add recommendation on how would you fix the workflow?

I expected the AI will pick up the error immediately and proposed a fix, additionally, it might give feature suggestion such as setting evidence and comment field as required before resolving an issue, re-opening an issue workflow, or even issue linking.

I would reject evidence required, reopening issue, and linking issue at these are extension features. A bare minimum would be a comment is required before resolving an issue.

4 Adapt to a new constraint

When offline, the shared device can show the last synced issue count from local storage. For example, Staff A acknowledges an issue offline while Manager B resolves it online. If the older update uploads later, it may overwrite the current status and make the dashboard incorrect.

The server should remain the source of truth. On reconnect, refresh from the server before syncing. Apply the offline update only if the issue is unchanged; otherwise, keep the server version and show a conflict.

I would track open issues. This may rise because acknowledged issues remain visible, even though the data is more accurate.

Time spent and AI use

Approximate time spent: 20 minutes. I used OpenAI Codex to review my draft, correct grammar, and suggest scenario for Q4. I answered Q1, Q2, Q3 with my own answers