NAKISA®

Software Developer Take-Home Challenge

VERSION 04.2025

Prepared by **The R&D Team @Nakisa**

APRIL 2025

Table of Contents

Guidelines

Welcome to the Nakisa Software Developer Take-Home Challenge – Finance Team. This assessment is designed to simulate a real-world technical task in our engineering environment. It focuses on backend and frontend development using Java (Spring Boot, JPA) and Vue.js.

You will enhance and test a pre-built Expense Tracker application. This challenge is structured to help us understand how you approach coding problems, organize your work, and apply clean coding principles.

⏱ Timeline

- You will have 48 hours to complete and submit your assignment after receiving this file.
- Please submit your completed code and presentation directly via the file uploader in CoderPad.

What's Included

- Backend: Spring Boot application (Java, JPA).
- Frontend: Vue.js application.
- Setup instructions, four exercises, and a final deliverable checklist.

Evaluation Criteria

- Implementation quality & completeness.
- Clean, maintainable code (naming, structure, design).

- Appropriate use of object-oriented principles.
- Test coverage (unit & integration).
- Frontend functionality and responsiveness.
- Overall problem-solving approach.

Use of AI Tools

We understand that you may choose to leverage tools such as GitHub Copilot or ChatGPT. While we welcome modern tools to streamline your workflow, your final submission must reflect your own logic, approach, and understanding. We will be reviewing for authenticity and thoughtful execution.

Deliverables

1. Your completed code (zipped or GitHub repo link).
2. A brief PowerPoint (4–6 slides) summarizing:
    a. Your approach to each exercise.
    b. Key challenges and how you solved them.
    c. Suggestions for improving scalability or performance.
3. A 20–25-minute presentation via screen-sharing (we'll coordinate the time with you).

We thank you again for your interest in Nakisa and look forward to seeing how you approach this challenge. If you have questions, feel free to reach out before your time limit expires. Good luck!

Nakisa Software Developer Take-Home Challenge

🚀 TEST OVERVIEW

You will work on a pre-existing **Expense Tracker** application with the following tech stack:

- **Backend:** Java (Spring Boot), JPA
- **Frontend:** Vue.js

The current application supports basic expense management. Your challenge is to complete a series of tasks that simulate real-world feature enhancements and improvements.

🧩 PROJECT SETUP

1. Backend

- Navigate to the backend/ directory.
- Run mvn spring-boot:run
- App will run on http://localhost:8080

2. Frontend

- Navigate to the frontend/ directory.
- Run npm install to install dependencies.
- Run npm run serve
- App typically runs on http://localhost:3000

Exercises

EXERCISE 1: IMPLEMENT ADVANCED FILTERING AND PAGINATION

Backend:

- Create a new endpoint: GET /api/expenses
- Add support for filtering by:
    - Category
    - Date range
    - Amount (greater than / less than)
- Add pagination (page, size)
- Response should include:
    - Paginated data
    - Total number of pages and expenses

Frontend:

- Add UI controls for filtering (e.g., dropdowns, input fields)
- Add pagination UI (next, previous, page info)

EXERCISE 2: ADD VALIDATION AND ERROR HANDLING

Backend:

- Add validation to POST /api/expenses and PUT /api/expenses/{id}:
    - Amount must be > 0

    - Date must not be in the future

- Return 400 Bad Request with appropriate JSON error messages
- Use @ControllerAdvice for consistent error handling

Frontend:

- Display user-friendly error messages on form submission failure

---

EXERCISE 3: IMPLEMENT SOFT DELETE AND ARCHIVING

Backend:

- Modify DELETE /api/expenses/{id} to perform a soft delete using a deleted flag
- Create a new endpoint: GET /api/expenses/archived to return soft-deleted expenses
- Implement logic to automatically archive expenses older than 30 days (you may mock the date for testing)

Frontend:

- Change the “Delete” button to an “Archive” button
- Create a new section to view archived expenses

EXERCISE 4: WRITE COMPREHENSIVE TESTS

Backend:

- Unit test the ExpenseService using Mockito
- Cover scenarios with invalid inputs
- Add at least one integration test for ExpenseController using @SpringBootTest

Frontend:

- Write a test for ExpenseForm.vue using Vue Test Utils
- Ensure it emits the correct event when an expense is added

Submission Instructions

Please upload the following directly through the CoderPad interface:

1. **Updated codebase** (zipped or GitHub repo)
2. **PowerPoint presentation (4–6 slides)** including:
    - Summary of your implementation and decisions
    - Any challenges you faced and how you solved them
    - One suggestion for improving scalability or performance (e.g., caching, architecture, etc.)
    - One additional feature or change in design you recommend.

Presentation

In your 20–25-minute screen-sharing session, we'll ask you to:

- Walk us through your code and decisions.
- Explain your testing strategy.
- Discuss your scalability improvement suggestion.
- Answer questions about trade-offs and edge cases.

THIS PAGE LEFT INTENTIONALLY BLANK

NAKISA®