Readme

Collaborative Task Management Backend Technical Setup & Testing Guide

Prepared by: *Neharika Sharma*

Date: *September 1st 2025*

Project Overview

This is a **Spring Boot (Java 17)** backend for a collaborative task manager. Teams can:

- Create **boards** with multiple **lists** (To-Do, In Progress, Done).
- CRUD **tasks** with title, description, due date, and status.
- **Assign** tasks to users and track progress via an **activity feed**.
- **Real-time updates** via **SSE (Server-Sent Events)**.
- **Email** notifications for assignment + **scheduled** due-soon reminders.
- **Analytics** for admins (task counts, most active users, avg completion time).

Key Features

- **Auth**: JWT-based signup/login.
- **Authorization**: Global roles (**USER**, **ADMIN**) + per-board roles (**OWNER**, **MEMBER**, **VIEWER**).
- **Persistence**: PostgreSQL (prod) / H2 in-memory (dev).
- **Hardening**: Rate limiting, HTML sanitization, centralized errors.
- **Performance**: Spring Cache (in-memory by default; Redis-ready).

Quick Links

- Swagger UI: http://localhost:8080/swagger-ui/index.html
- MailHog UI (dev email inbox): http://localhost:8025
- H2 Console (dev profile): http://localhost:8080/h2-console/
  JDBC: jdbc:h2:mem:taskdb | User: sa | Password: *(blank)*

Prerequisites

- **Java 17+**
- **Maven 3.9+** (wrapper ./mvnw included)
- **Docker** (for Postgres, Redis, MailHog)
- **curl** or Postman for API testing

Ports used locally: **8080** (app), **5432** (Postgres), **6379** (Redis, optional), **1025/8025** (MailHog).

Setup & Testing Guide

Database Setup & Running Locally

Option A: Quick Dev Profile (in-memory H2)

```
cd ~/Desktop/task-backend

SPRING_PROFILES_ACTIVE=dev ./mvnw spring-boot:run
```

- Runs against **H2** in-memory DB.

- Use http://localhost:8080/h2-console/ (JDBC URL: jdbc:h2:mem:taskdb, user: sa, no password) to inspect DB.

Option B: Full Setup with Docker (Postgres + Redis + MailHog)

```
docker run -d --name task_db -e POSTGRES_PASSWORD=secret -p 5432:5432 postgres:16

docker run -d --name redis -p 6379:6379 redis:7

docker run -d --name mailhog -p 1025:1025 -p 8025:8025 mailhog/mailhog
```

Then run:

```
./mvnw spring-boot:run
```

2. Running Tests

Run the unit/integration tests:

```
./mvnw test
```

- Reports are in target/surefire-reports/.
- To run one class only:

```
./mvnw -Dtest=SomeTestClass test
```

3. Manual API Testing (Stage 1–4)

These curl scripts validate every requirement in your project brief.

Always run **Terminal A** with the server, and use **Terminal B** for API calls.

Stage 1 – Core API & Data Model

Terminal A – start the app

```
cd ~/Desktop/task-backend
SPRING_PROFILES_ACTIVE=dev ./mvnw spring-boot:run
```

Terminal B – API walkthrough

0) Setup

```
BASE="http://localhost:8080"
CT="Content-Type: application/json"
```

1) Auth: signup / login / me

- Signs up Alice, logs in, gets JWT, and verifies /me.

```
TOKEN=$(
```

```
  curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
    -d '{"email":"alice@example.com","password":"secret123","name":"Alice"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" \
    -d '{"email":"alice@example.com","password":"secret123"}' | jq -r .token
)
AUTH="Authorization: Bearer $TOKEN"

curl -i "$BASE/api/auth/me" | sed -n '1,8p'
curl -s "$BASE/api/auth/me" -H "$AUTH" | jq .
```

2) Boards: create, list, get, rename

- Creates board, fetches it, renames it.

```
BOARD_ID=$(curl -s "$BASE/api/boards" -H "$AUTH" -H "$CT" -d '{"name":"Demo Board"}'
| jq -r .id)
curl -s "$BASE/api/boards/$BOARD_ID" -H "$AUTH" | jq .
curl -s -X PUT "$BASE/api/boards/$BOARD_ID" -H "$AUTH" -H "$CT" \
  -d '{"name":"Demo Board (renamed)"}' | jq .
```

3) Lists: create, update

```
TODO_ID=$(curl -s "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH" -H "$CT" \
  -d '{"name":"To Do","position":1}' | jq -r .id)
DOING_ID=$(curl -s "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH" -H "$CT" \
  -d '{"name":"In Progress","position":2}' | jq -r .id)

curl -s -X PUT "$BASE/api/boards/$BOARD_ID/lists/$DOING_ID" -H "$AUTH" -H "$CT" \
  -d '{"name":"Doing","position":2}' | jq .
```

4) Tasks: CRUD + assign

- Creates a task, updates it, patches status, assigns to Alice, then deletes.

```
TASK_ID=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH" -H "$CT" \
  -d '{"title":"Write tests","description":"Stage 1 check","dueDate":"2030-01-01"}' | jq -r .id)

curl -s -X PATCH "$BASE/api/tasks/$TASK_ID" -H "$AUTH" -H "$CT" -d '{"status":"DONE"}' |
jq .

ALICE_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH" | jq -r .userId)
curl -s -X PUT "$BASE/api/tasks/$TASK_ID/assignees" -H "$AUTH" -H "$CT" \
  -d "{\"userIds\":[$ALICE_ID]}" | jq .

curl -i -s -X DELETE "$BASE/api/lists/$TODO_ID/tasks/$TASK_ID" -H "$AUTH" | sed -n
'1,4p'
```

5) Access Control

- A non-member (Bob) tries to fetch Alice's board → should be **403 Forbidden**.

```
TOKEN_B=$(curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
  -d '{"email":"bob@example.com","password":"secret123","name":"Bob"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" -d
'{"email":"bob@example.com","password":"secret123"}' | jq -r .token)
AUTH_B="Authorization: Bearer $TOKEN_B"

curl -i -s "$BASE/api/boards/$BOARD_ID" -H "$AUTH_B" | sed -n '1,8p'
```

Stage 2 – Collaboration & Access Control

- Create 3 users (Alice=Owner, Bob=Editor, Carol=Viewer).
- Validate **permissions** (Bob can edit, Carol blocked).
- Check **activity log**.
- Run **search & filters** queries with pagination.

0) Setup

```
BASE="http://localhost:8080"
CT="Content-Type: application/json"
```

1) Create three users (roles on the board will differ later)

- **Alice** will own the board
- **Bob** will be the **Editor** (MEMBER)
- **Carol** will be the **Viewer**

```
TOKEN_A=$(
  curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
    -d '{"email":"alice@example.com","password":"secret123","name":"Alice"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" \
    -d '{"email":"alice@example.com","password":"secret123"}' | jq -r .token
)
AUTH_A="Authorization: Bearer $TOKEN_A"
ALICE_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH_A" | jq -r .userId)



TOKEN_B=$(
  curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
    -d '{"email":"bob@example.com","password":"secret123","name":"Bob"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" \
    -d '{"email":"bob@example.com","password":"secret123"}' | jq -r .token
)
AUTH_B="Authorization: Bearer $TOKEN_B"
```

```
BOB_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH_B" | jq -r .userId)


TOKEN_C=$(
  curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
    -d '{"email":"carol@example.com","password":"secret123","name":"Carol"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" \
    -d '{"email":"carol@example.com","password":"secret123"}' | jq -r .token
)
AUTH_C="Authorization: Bearer $TOKEN_C"
CAROL_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH_C" | jq -r .userId)

echo "ALICE=$ALICE_ID  BOB=$BOB_ID  CAROL=$CAROL_ID"
```

2) Board membership & permissions

2.1 Alice creates a board (she is the Owner by default)

```
BOARD_ID=$(curl -s "$BASE/api/boards" -H "$AUTH_A" -H "$CT" -d '{"name":"Stage2
Board"}' | jq -r .id)
echo "BOARD_ID=$BOARD_ID"
```

2.2 Non-member access is blocked (Bob should receive 403)

```
curl -i -s "$BASE/api/boards/$BOARD_ID" -H "$AUTH_B" | sed -n '1,8p'
```

2.3 Alice invites Bob (Editor) and Carol (Viewer)

```
curl -s -X POST
"$BASE/api/boards/$BOARD_ID/members?userId=$BOB_ID&role=MEMBER" -H
"$AUTH_A" | jq .

curl -s -X POST
"$BASE/api/boards/$BOARD_ID/members?userId=$CAROL_ID&role=VIEWER" -H
"$AUTH_A" | jq .

curl -s "$BASE/api/boards/$BOARD_ID/members" -H "$AUTH_A" | jq .
```

2.4 Permission checks

- **Viewer (Carol)** cannot create lists → expect **403**
- **Editor (Bob)** can create lists → expect **200 + JSON**
- **Editor** cannot rename board (owner-only) → expect **403**
- **Owner (Alice)** can rename
- **Owner** can remove members

```
curl -i -s -X POST "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH_C" -H "$CT" \
  -d '{"name":"Viewer List","position":1}' | sed -n '1,10p'

TODO_ID=$(curl -s -X POST "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH_B" -H "$CT" \
  -d '{"name":"To Do","position":1}' | jq -r .id)
DOING_ID=$(curl -s -X POST "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH_B" -H "$CT" \
  -d '{"name":"In Progress","position":2}' | jq -r .id)
echo "TODO=$TODO_ID DOING=$DOING_ID"

curl -i -s -X PUT "$BASE/api/boards/$BOARD_ID" -H "$AUTH_B" -H "$CT" \
  -d '{"name":"Hacked by Bob"}' | sed -n '1,10p'

curl -s -X PUT "$BASE/api/boards/$BOARD_ID" -H "$AUTH_A" -H "$CT" \
  -d '{"name":"Stage2 Board (official)"}' | jq .

curl -i -s -X DELETE "$BASE/api/boards/$BOARD_ID/members/$CAROL_ID" -H "$AUTH_A" | sed -n '1,6p'

curl -i -s "$BASE/api/boards/$BOARD_ID" -H "$AUTH_C" | sed -n '1,8p'
```

3) Activity logging

Create/update/delete tasks to generate entries, then fetch the board’s activity feed.

```
T1=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH_B" -H "$CT" \
  -d '{"title":"Spec review","description":"read docs","dueDate":"2030-01-01"}' | jq -r .id)
T2=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH_B" -H "$CT" \
```

```
  -d '{"title":"Write tests","description":"stage2 tests","dueDate":"2030-01-02"}' | jq -r .id)

curl -s -X PATCH "$BASE/api/tasks/$T2" -H "$AUTH_B" -H "$CT" -d '{"status":"DONE"}' | jq .

curl -i -s -X DELETE "$BASE/api/lists/$TODO_ID/tasks/$T1" -H "$AUTH_A" | sed -n '1,6p'

curl -s "$BASE/api/boards/$BOARD_ID/activity?page=0&size=20" -H "$AUTH_A" | jq .
```

4) Search & filters (with pagination)

Create a few tasks, assign one to Alice, and try various filter combinations (query text, status, assignee, due date range, and paging).

```
A=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH_B" -H "$CT" \
  -d '{"title":"Write docs","description":"user manual","dueDate":"2030-02-01"}' | jq -r .id)

B=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH_B" -H "$CT" \
  -d '{"title":"Fix bug 123","description":"null pointer","dueDate":"2030-01-15"}' | jq -r .id)

C=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH_B" -H "$CT" \
  -d '{"title":"Prepare demo","description":"slides + script","dueDate":"2030-01-20"}' | jq -r .id)

ALICE_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH_A" | jq -r .userId)
curl -s -X PUT "$BASE/api/tasks/$B/assignees" -H "$AUTH_A" -H "$CT" \
  -d "{\"userIds\":[$ALICE_ID]}" | jq . > /dev/null

curl -s -X PATCH "$BASE/api/tasks/$C" -H "$AUTH_B" -H "$CT" -d
'{"status":"IN_PROGRESS"}' | jq . > /dev/null
```

curl -s "$BASE/api/tasks/search?boardId=$BOARD_ID&q=write&page=0&size=10" -H "$AUTH_A" | jq .

curl -s "$BASE/api/tasks/search?boardId=$BOARD_ID&status=DONE&page=0&size=10" -H "$AUTH_A" | jq .

curl -s "$BASE/api/tasks/search?boardId=$BOARD_ID&assigneeId=$ALICE_ID&page=0&size=10" -H "$AUTH_A" | jq .

curl -s "$BASE/api/tasks/search?boardId=$BOARD_ID&from=2030-01-01&to=2030-01-31&page=0&size=10" -H "$AUTH_A" | jq .

curl -s "$BASE/api/tasks/search?boardId=$BOARD_ID&page=0&size=1" -H "$AUTH_A" | jq .

curl -s "$BASE/api/tasks/search?boardId=$BOARD_ID&page=1&size=1" -H "$AUTH_A" | jq .

5) Additional permission edge cases

Validate that non-owners cannot escalate roles or create resources they shouldn’t.

curl -i -s -X POST "$BASE/api/boards/$BOARD_ID/members?userId=$ALICE_ID&role=VIEWER" -H "$AUTH_B" | sed -n '1,8p'

curl -s -X POST "$BASE/api/boards/$BOARD_ID/members?userId=$CAROL_ID&role=VIEWER" -H "$AUTH_A" | jq .

curl -i -s -X POST "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH_C" -H "$CT" -d '{"name":"Nope","position":3}' | sed -n '1,8p'

curl -s "$BASE/api/boards/$BOARD_ID" -H "$AUTH_C" | jq .

Stage 3 – Real-time Updates & Notifications

This stage verifies:

- **Email notifications** when tasks are assigned (using MailHog locally).
- **Due-date reminder job** (scheduled task finds items due in 24h and emails assignees).
- **Real-time board events** via **Server-Sent Events (SSE)**.

You'll run the app, spin up Docker services (Postgres, MailHog, Redis), execute a test script, and finally watch live SSE updates while you mutate tasks.

Terminal A — Start the app (dev profile)

SPRING_PROFILES_ACTIVE=dev ./mvnw spring-boot:run

Terminal B — Ensure external services are running

Check Docker:

docker ps

If needed, (re)start containers:

docker run -d --name task_db -e POSTGRES_PASSWORD=secret -p 5432:5432 postgres:16 || true

docker rm -f mailhog 2>/dev/null || true
docker run -d --name mailhog -p 1025:1025 -p 8025:8025 mailhog/mailhog

docker rm -f redis 2>/dev/null || true
docker run -d --name redis -p 6379:6379 redis:7

Check Docker:

docker ps

Open the MailHog UI to view captured emails:

open http://localhost:8025 || xdg-open http://localhost:8025 || true

Terminal B — Run the Stage 3 test script

This script:

1. Creates users and a board, invites members.
2. Creates lists and a task, **assigns** it (triggers **assignment email**).

3. Creates a task due **tomorrow**, assigns it, waits for **scheduler** to send the **due-soon reminder**.
4. Prints MailHog counts and recent messages for easy verification.

```
cat > stage3-test.sh <<'SH'
set -euo pipefail

BASE="${BASE:-http://localhost:8080}"
MAILHOG_API="${MAILHOG_API:-http://localhost:8025/api/v2/messages}"
CT="Content-Type: application/json"

have() { command -v "$1" >/dev/null 2>&1; }
need() { if ! have "$1"; then echo "Missing required tool: $1" >&2; exit 1; fi; }
need curl; need jq

title() { echo; echo "=== $* ==="; }
mh_count() { curl -s "$MAILHOG_API" | jq -r '.total // 0'; }
mh_last_n() {
  local n="${1:-5}"
  curl -s "$MAILHOG_API" \
    | jq -r --argjson N "$n" '
        .items[0:$N] |
        map({
          subj: (.Content.Headers.Subject[0] // ""),
          to:   (.Content.Headers.To[0] // ""),
          from: (.Content.Headers.From[0] // ""),
          date: (.Created // "")
        })'
}

tomorrow_ymd() {
  if date -u -v+1d +"%Y-%m-%d" >/dev/null 2>&1; then
    date -u -v+1d +"%Y-%m-%d"
  else
    python3 - <<'PY'
from datetime import datetime, timedelta, timezone
print((datetime.now(timezone.utc)+timedelta(days=1)).strftime("%Y-%m-%d"))
PY
  fi
}

title "Signing up / logging in users"
TOKEN_A=$(curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
  -d '{"email":"alice@example.com","password":"secret123","name":"Alice"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" -d
'{"email":"alice@example.com","password":"secret123"}' | jq -r .token)
AUTH_A="Authorization: Bearer $TOKEN_A"
```

```
TOKEN_B=$(curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
  -d '{"email":"bob@example.com","password":"secret123","name":"Bob"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" -d
'{"email":"bob@example.com","password":"secret123"}' | jq -r .token)
AUTH_B="Authorization: Bearer $TOKEN_B"

TOKEN_C=$(curl -sS -f "$BASE/api/auth/signup" -H "$CT" \
  -d '{"email":"carol@example.com","password":"secret123","name":"Carol"}' | jq -r .token
2>/dev/null \
  || curl -sS -f "$BASE/api/auth/login" -H "$CT" -d
'{"email":"carol@example.com","password":"secret123"}' | jq -r .token)
AUTH_C="Authorization: Bearer $TOKEN_C"

ALICE_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH_A" | jq -r .userId)
BOB_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH_B" | jq -r .userId)
CAROL_ID=$(curl -s "$BASE/api/auth/me" -H "$AUTH_C" | jq -r .userId)
echo "ALICE=$ALICE_ID  BOB=$BOB_ID  CAROL=$CAROL_ID"

title "Create board and memberships"
BOARD_ID=$(curl -s "$BASE/api/boards" -H "$AUTH_A" -H "$CT" -d '{"name":"Stage3
Board"}' | jq -r .id)
echo "BOARD_ID=$BOARD_ID"
curl -s -X POST
"$BASE/api/boards/$BOARD_ID/members?userId=$BOB_ID&role=MEMBER" -H
"$AUTH_A" > /dev/null
curl -s -X POST
"$BASE/api/boards/$BOARD_ID/members?userId=$CAROL_ID&role=VIEWER" -H
"$AUTH_A" > /dev/null
echo "Added Bob as MEMBER, Carol as VIEWER"

title "Create lists and a task"
TODO_ID=$(curl -s "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH_A" -H "$CT" -d
'{"name":"To Do","position":1}' | jq -r .id)
DOING_ID=$(curl -s "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH_A" -H "$CT" -d
'{"name":"In Progress","position":2}' | jq -r .id)
echo "TODO_ID=$TODO_ID  DOING_ID=$DOING_ID"

TASK_ID=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH_A" -H "$CT" -d
'{"title":"Stage3 seed","description":"for RT/email tests","dueDate":"2030-01-01"}' | jq -r .id)
echo "TASK_ID=$TASK_ID"

title "Assignment email test (MailHog should receive 1 email to bob@example.com)"
BEFORE=$(mh_count || echo 0)
echo "MailHog messages before: $BEFORE"
curl -s -X PUT "$BASE/api/tasks/$TASK_ID/assignees" -H "$AUTH_A" -H "$CT" -d
"{\"userIds\":[$BOB_ID]}" > /dev/null
```

```
sleep 2
AFTER=$(mh_count || echo 0)
echo "MailHog messages after:  $AFTER"
echo "Last messages:"; mh_last_n 3
if [ "$AFTER" -le "$BEFORE" ]; then
  echo "WARNING: MailHog count did not increase. Check spring.mail.* config and MailHog
on localhost:1025" >&2
fi

title "Due-soon reminder test (create task due tomorrow and assign Bob)"
TOMORROW=$(tomorrow_ymd)
echo "Tomorrow (UTC) = $TOMORROW"
REM_TASK_ID=$(curl -s "$BASE/api/lists/$TODO_ID/tasks" -H "$AUTH_A" -H "$CT" \
  -d "{\"title\":\"Due soon\",\"description\":\"reminder test\",\"dueDate\":\"$TOMORROW\"}" | jq
-r .id)
curl -s -X PUT "$BASE/api/tasks/$REM_TASK_ID/assignees" -H "$AUTH_A" -H "$CT" -d
"{\"userIds\":[$BOB_ID]}" > /dev/null

echo "Waiting 12s for scheduler (adjust if your @Scheduled frequency is longer)..."
sleep 12

AFTER2=$(mh_count || echo 0)
echo "MailHog messages now:   $AFTER2"
echo "Last messages:"; mh_last_n 5

echo
echo "DONE. Next: open a realtime stream (SSE or WS) in another terminal and then
mutate tasks here to see events."
SH
chmod +x stage3-test.sh
./stage3-test.sh
```

Terminal C — Subscribe to realtime updates (SSE)

```
BASE="http://localhost:8080"

CT="Content-Type: application/json"



TOKEN_A=$(curl -sS -f "$BASE/api/auth/login" -H "$CT" -d
'{"email":"alice@example.com","password":"secret123"}' | jq -r .token)

AUTH_A="Authorization: Bearer $TOKEN_A"
```

```
BOARD_ID=$(curl -s "$BASE/api/boards" -H "$AUTH_A" | jq -r '.[0].id')

echo "BOARD_ID=$BOARD_ID"


curl -N -H "$AUTH_A" "$BASE/api/boards/$BOARD_ID/stream"
```

Terminal B — Trigger real-time events

With Terminal C streaming, run a few mutations and watch events flow in:

```
curl -s "$BASE/api/lists/1/tasks" -H "$AUTH_A" -H "$CT" \
  -d '{"title":"RT check","description":"sse","dueDate":"2030-03-01"}' | jq .

TASK_ID=2
curl -s -X PATCH "$BASE/api/tasks/$TASK_ID" -H "$AUTH_A" -H "$CT" -d
'{"status":"IN_PROGRESS"}' | jq .

curl -s -X PUT "$BASE/api/boards/$BOARD_ID" -H "$AUTH_A" -H "$CT" \
  -d '{"name":"Stage3 Board (rt)"}' | jq .
```

Expected: Terminal C displays SSE events like task-changed and board updates, confirming the real-time pipeline works.

Stage 4 — Performance, Security & Analytics

This stage verifies:

- **Caching** behavior (warm vs cold responses).
- **Security hardening**: input validation/sanitization + rate limiting.
- **Admin analytics** endpoints after elevating a user to **ADMIN**.

Terminal A — Start the app (dev profile)

```
cd ~/Desktop/task-backend
SPRING_PROFILES_ACTIVE=dev ./mvnw spring-boot:run
```

Terminal B — Env, seed data, and baseline entities

```
export BASE="http://localhost:8080"
export CT="Content-Type: application/json"

curl -s -f "$BASE/api/auth/signup" -H "$CT" -d
'{"email":"alice@example.com","password":"secret123","name":"Alice"}' >/dev/null || true
```

```
TOKEN_A=$(curl -s -f "$BASE/api/auth/login" -H "$CT" -d
'{"email":"alice@example.com","password":"secret123"}' | jq -r .token)
export AUTH_A="Authorization: Bearer $TOKEN_A"

BOARD_ID=$(curl -s -X POST "$BASE/api/boards" -H "$AUTH_A" -H "$CT" -d
'{"name":"Stage4 Board"}' | jq -r .id)
export BOARD_ID

LIST_ID=$(curl -s -X POST "$BASE/api/boards/$BOARD_ID/lists" -H "$AUTH_A" -H "$CT"
-d '{"name":"To Do","position":1}' | jq -r .id)
export LIST_ID

TASK_ID=$(curl -s -X POST "$BASE/api/lists/$LIST_ID/tasks" -H "$AUTH_A" -H "$CT" -d
'{"title":"Baseline task","description":"ok","dueDate":"2030-01-01"}' | jq -r .id)
export TASK_ID
```

Cache check — cold vs warm responses

```
cat > cache-test.sh <<'SH'
set -euo pipefail
: "${BASE:=http://localhost:8080}"
: "${AUTH_A:?missing AUTH_A}"
: "${BOARD_ID:?missing BOARD_ID}"
curl -s -o /dev/null -w "cold: %{time_total}\n"  -H "$AUTH_A"
"$BASE/api/boards/$BOARD_ID"
curl -s -o /dev/null -w "warm1: %{time_total}\n" -H "$AUTH_A"
"$BASE/api/boards/$BOARD_ID"
curl -s -o /dev/null -w "warm2: %{time_total}\n" -H "$AUTH_A"
"$BASE/api/boards/$BOARD_ID"
SH
chmod +x cache-test.sh
./cache-test.sh
```

Rename the board (which should invalidate any cache) and re-run:

```
curl -s -X PUT "$BASE/api/boards/$BOARD_ID" -H "$AUTH_A" -H "$CT" -d
'{"name":"Stage4 Board cache test"}' >/dev/null
./cache-test.sh
```

Security hardening checks

1. **Invalid enum value** for status → expect **400 Bad Request** with a helpful error:

curl -i -s -X PATCH "$BASE/api/tasks/$TASK_ID" -H "$AUTH_A" -H "$CT" -d '{"status":"NOT_A_STATUS"}' | sed -n '1,25p'

2. **HTML sanitization** on task title (script tags removed):

   BAD_ID=$(curl -s "$BASE/api/lists/$LIST_ID/tasks" -H "$AUTH_A" -H "$CT" -d '{"title":"<script>alert(1)</script>","description":"xss","dueDate":"2030-01-01"}' | jq -r .id)

   curl -s -H "$AUTH_A" "$BASE/api/lists/$LIST_ID/tasks" | jq .

3. **Oversized payload** (truncation and/or 400 depending on validators):

   python3 - <<'PY' | curl -i -s "$BASE/api/lists/$LIST_ID/tasks" -H "$AUTH_A" -H "$CT" -d @- | sed -n '1,25p'

   print('{"title":"' + "A"*10000 + '","description":"too long","dueDate":"2030-01-01"}')

   PY

4. **Rate limiting** (burst 150 requests; expect many **200** plus some **429**):

   cat > rate-burst.sh <<'SH'

   set -euo pipefail

   : "${BASE:=http://localhost:8080}"

   : "${AUTH_A:?missing AUTH_A}"

   : "${BOARD_ID:?missing BOARD_ID}"

   seq 1 150 | xargs -n1 -P16 -I{} curl -s -o /dev/null -w "%{http_code}\n" -H "$AUTH_A" "$BASE/api/boards/$BOARD_ID" | sort | uniq -c

   SH

   chmod +x rate-burst.sh

   ./rate-burst.sh

Elevate to ADMIN and test analytics

Use the **H2 Console** (dev) to promote Alice:

1. Open: http://localhost:8080/h2-console/
2. Fill:
   - **JDBC URL:** jdbc:h2:mem:taskdb
   - **User:** sa
   - **Password:** *(leave blank)*
3. Connect and run:

UPDATE users SET role='ADMIN' WHERE email='alice@example.com';

Now call admin endpoints:

```
TOKEN_ADMIN=$(curl -s -f "$BASE/api/auth/login" -H "$CT" -d
'{"email":"alice@example.com","password":"secret123"}' | jq -r .token)

export AUTH_ADMIN="Authorization: Bearer $TOKEN_ADMIN"

curl -s -H "$AUTH_ADMIN" "$BASE/api/admin/analytics/board-task-counts" | jq .

curl -s -H "$AUTH_ADMIN" "$BASE/api/admin/analytics/avg-completion-per-board" |
jq .

curl -s -H "$AUTH_ADMIN" "$BASE/api/admin/analytics/most-active-users?limit=5" |
jq .
```

Seed a bit more data to make analytics interesting, then re-run:

```
curl -s -X POST "$BASE/api/lists/$LIST_ID/tasks" -H "$AUTH_A" -H "$CT" -d
'{"title":"Finish docs","description":"x","dueDate":"2030-01-02"}' >/dev/null

curl -s -X POST "$BASE/api/lists/$LIST_ID/tasks" -H "$AUTH_A" -H "$CT" -d
'{"title":"Ship v1","description":"y","dueDate":"2030-01-03"}' >/dev/null

curl -s -X PATCH "$BASE/api/tasks/$TASK_ID" -H "$AUTH_A" -H "$CT" -d
'{"status":"DONE"}' >/dev/null


curl -s -H "$AUTH_ADMIN" "$BASE/api/admin/analytics/board-task-counts" | jq .

curl -s -H "$AUTH_ADMIN" "$BASE/api/admin/analytics/avg-completion-per-board" |
jq .
```

Troubleshooting;

- 403 on authenticated routes: make sure you're passing `Authorization: Bearer <token>`.

- MailHog not receiving emails: confirm containers (`docker ps`) and that app is using `spring.mail.host=localhost, port=1025`.

- H2 Console login fails: profile must be `dev`; URL `jdbc:h2:mem:taskdb`, user `sa`, blank password.

- No 429 in rate test: RateLimitFilter window/max can be tuned with env vars:

- `RATELIMIT_WINDOW_MS=60000` and `RATELIMIT_MAX=100`

- CORS from a frontend at `http://localhost:3000`: already allowed by default in `SecurityConfig`.