Tech Doc

Collaborative Task Management Backend Technical Documentation

Prepared by: *Neharika Sharma*

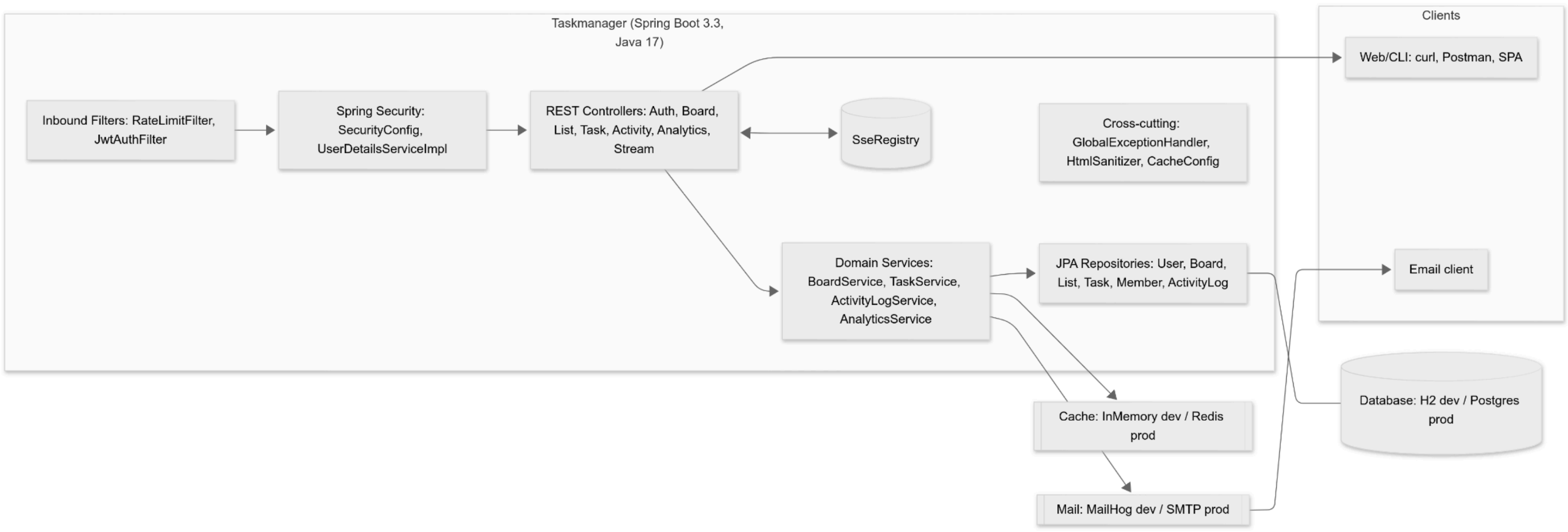
Date: *September 1st 2025*

Deliverables in this Document:

- System Architecture
- ER Diagram of the Database
- API Endpoint List (with request/response examples)
- Authentication and Role-Based Access Control

System architecture;

Runtime topology


Taskmanager (Spring Boot 3.3, Java 17)
Inbound Filters: RateLimitFilter, JwtAuthFilter
Spring Security: SecurityConfig, UserDetailsServiceImpl
REST Controllers: Auth, Board, List, Task, Activity, Analytics, Stream
SseRegistry
Cross-cutting: GlobalExceptionHandler, HtmlSanitizer, CacheConfig
Domain Services: BoardService, TaskService, ActivityLogService, AnalyticsService
JPA Repositories: User, Board, List, Task, Member, ActivityLog
Clients
Web/CLI: curl, Postman, SPA
Email client
Database: H2 dev / Postgres prod
Cache: InMemory dev / Redis prod
Mail: MailHog dev / SMTP prod

* You're using ConcurrentMapCacheManager today; switching to Redis is a drop-in via CacheManager in CacheConfig.

Layered (hexagonal-ish) breakdown

Presentation / Edge

- **Filters:** RateLimitFilter (simple IP window counter) → JwtAuthFilter (extract/verify JWT, set SecurityContext).
- **Security config:** SecurityConfig (stateless session, CORS, endpoint rules, provider chain).
- **Controllers (ports):**
    - AuthController, BoardController, BoardListController, TaskController, TaskAssignmentController, TaskStatusController, TaskSearchController, ActivityController, AdminAnalyticsController, StreamController (SSE).

Domain / Application

- **Services (use cases):**
    - AuthService (signup/login, token issue via JwtService).
    - BoardService + BoardMembershipService (board CRUD, membership, cache invalidate).
    - TaskService (CRUD, assign, status change, search composition → TaskSpecifications). Side-effects encapsulated with *best-effort* try/catch: ActivityLogService (+log), SseRegistry (+notify), MailService (on assignment), keeping main transaction happy.
    - AnalyticsService (read-only JPQL over Task/Board/Activity).
    - DueDateReminderJob (scheduled polling; finds tasks due in next 24h and emails assignees).

    - SseRegistry (keeps SseEmitters per board).
- **Cross-cutting:**
    - GlobalExceptionHandler (uniform JSON for 400/403/404/500).
    - HtmlSanitizer (strips script/handlers, trims long inputs).
    - CacheConfig (declares caches: boardDetails, userProfiles, searchResults).

Data / Persistence (adapters)

- **Entities:** User, Board, BoardList, Task, ActivityLog, plus join/role models BoardMember, BoardMembership, TaskAssignee, enums Role, BoardRole, TaskStatus.
- **Repositories:** Spring Data JPA interfaces for each aggregate root (UserRepository, BoardRepository, BoardListRepository, TaskRepository, ActivityLogRepository, …).

Request lifecycle (what happens to a call)

HTTP request in → RateLimitFilter (429 if window exceeded).

→ JwtAuthFilter (validate JWT via JwtService, build UsernamePasswordAuthenticationToken).

→ Spring Security authz (from SecurityConfig rules + @PreAuthorize on controllers).

→ Controller validates DTO (@Valid) & calls Service.

→ Service does transactional work with Repos; applies cache (@Cacheable, @CacheEvict), sanitizes free text (HtmlSanitizer).

→ Service triggers non-critical side-effects (activity log, SSE, email) in try/catch so failures don’t break the main mutation.

→ Controller returns DTO; GlobalExceptionHandler ensures consistent errors.

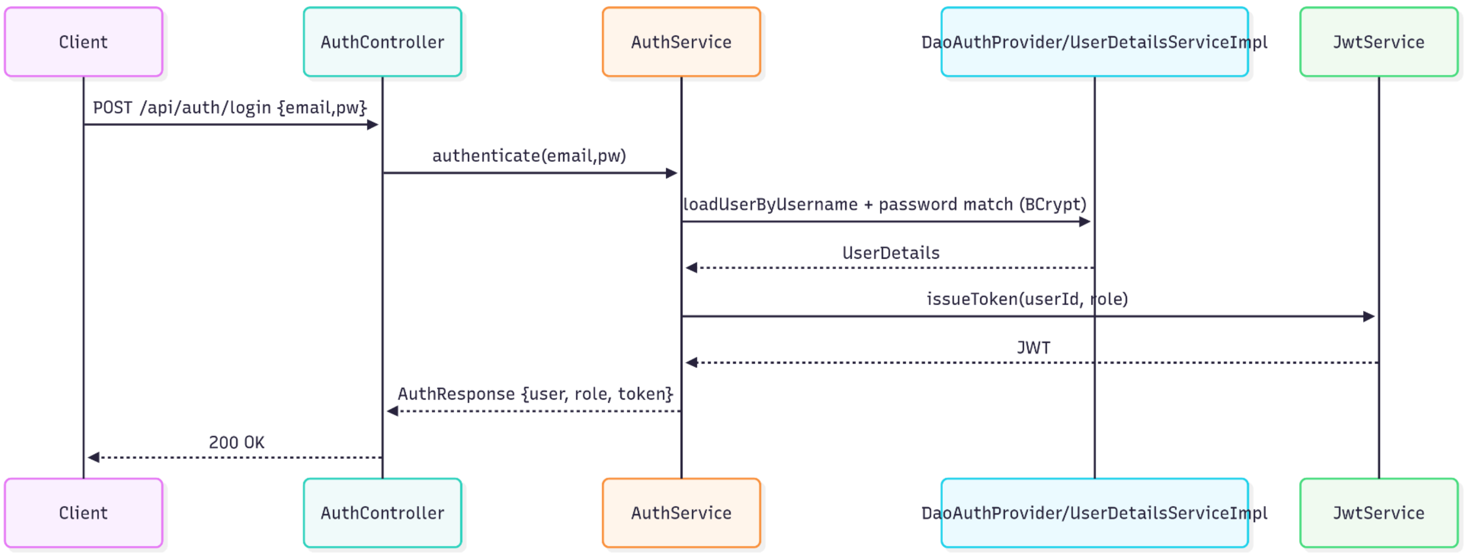
Detailed component map ←→ code

- **Auth**
    - AuthController, AuthService, DTOs (SignupRequest, LoginRequest, AuthResponse).
    - JwtService, UserDetailsServiceImpl, SecurityConfig (permits /api/auth/**, guards /api/admin/**).
- **Boards/Lists**
    - BoardController, BoardService, BoardRepository, DTOs (BoardDto, BoardWithListsDto, ListDto, …).
    - **Membership:** BoardMembershipController, BoardMembershipService, BoardMemberRepository, enums BoardRole (OWNER/MEMBER/VIEWER).
    - **Caching:** BoardService#getBoard(...) uses @Cacheable("boardDetails"); update, membershipChange use @CacheEvict.

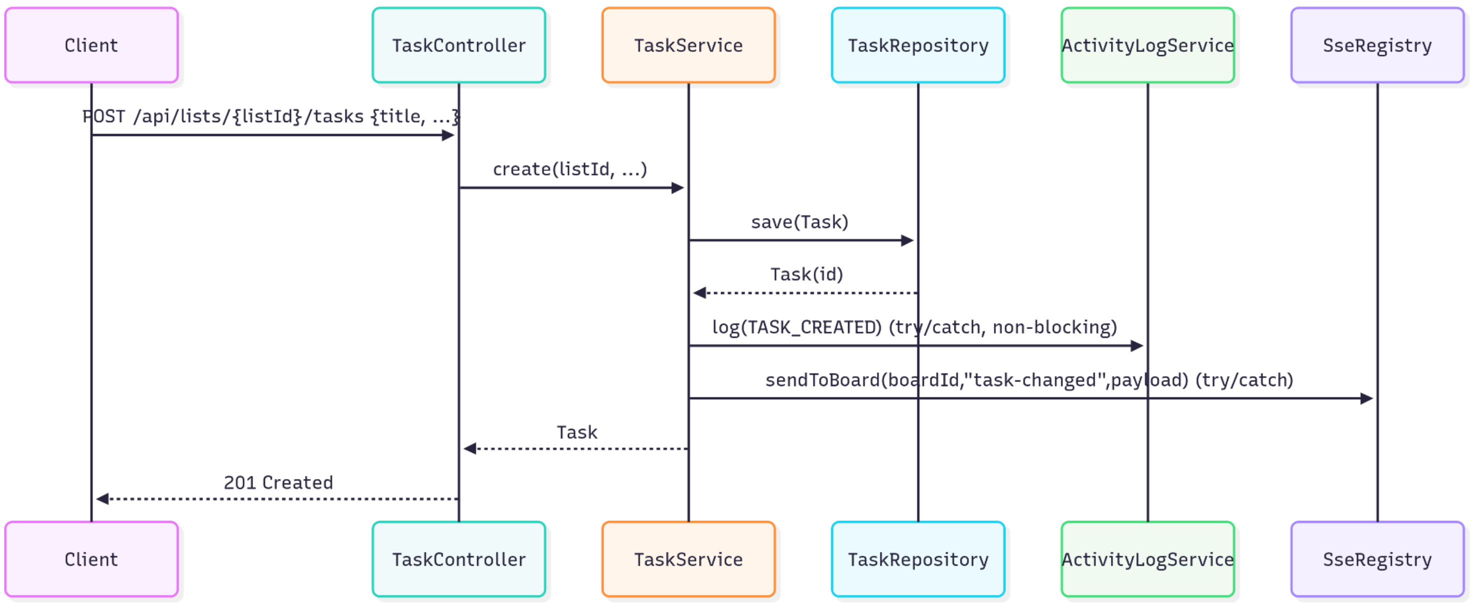
- **Tasks**
    - TaskController, TaskAssignmentController, TaskStatusController, TaskService, TaskRepository.
    - TaskSpecifications build dynamic Specification<Task> for search (TaskSearchController).
    - **Side-effects on create/update/delete/assign:** ActivityLogService, SseRegistry, MailService.
- **Activity**
    - ActivityLogService, ActivityController, ActivityLogRepository, enum ActivityType.
- **Realtime**
    - StreamController (SSE endpoint per board), SseRegistry holds emitters in memory keyed by board id.
- **Notifications**
    - MailService (Spring Mail); DueDateReminderJob (@Scheduled, window next 24h).
- **Analytics (admin)**
    - **AnalyticsService uses JPQL (portable) for:**
        - Board status counts
        - Avg completion hours per board
        - Most active users (by activity log count)
    - AdminAnalyticsController guarded by @PreAuthorize("hasRole('ADMIN')")

Key sequences

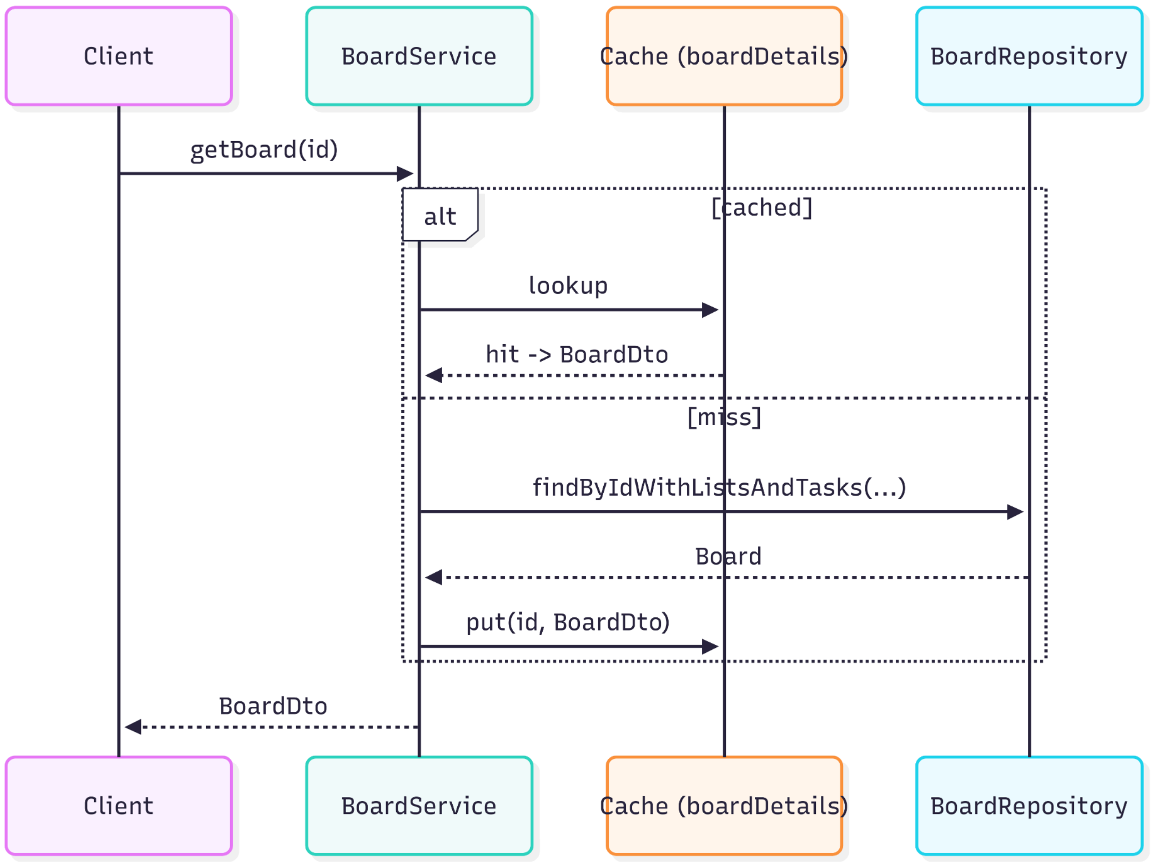
a) Login (JWT)


Client
AuthController
AuthService
DaoAuthProvider/UserDetailsServiceImpl
JwtService
POST /api/auth/login {email,pw}
authenticate(email,pw)
loadUserByUsername + password match (BCrypt)
UserDetails
issueToken(userId, role)
JWT
AuthResponse {user, role, token}
200 OK

b) Create task with side-effects


Client
TaskController
TaskService
TaskRepository
ActivityLogService
SseRegistry
POST /api/lists/{listId}/tasks {title, ...}
create(listId, ...)
save(Task)
Task(id)
log(TASK_CREATED) (try/catch, non-blocking)
sendToBoard(boardId,"task-changed",payload) (try/catch)
Task
201 Created

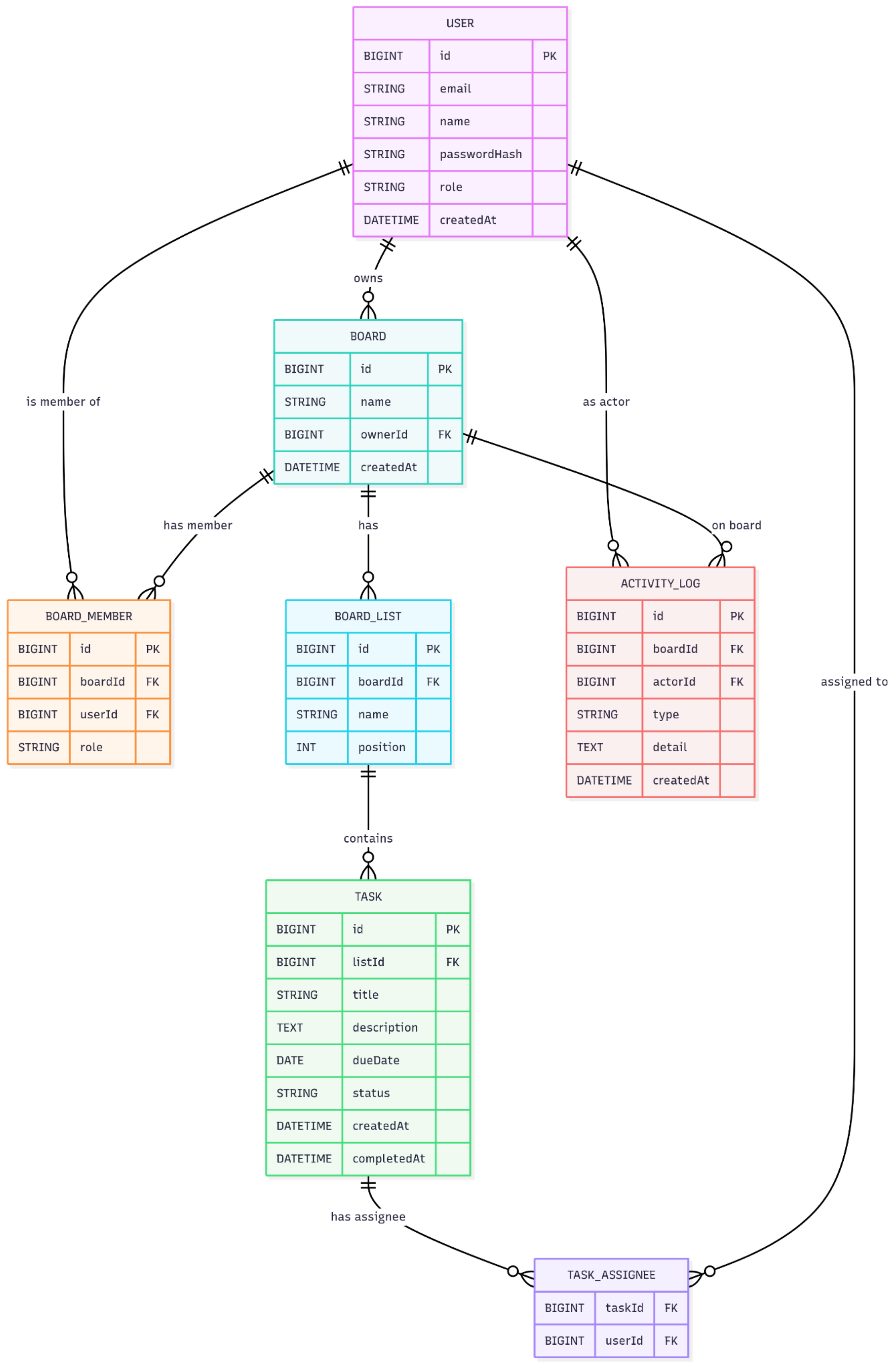
c) Board read with cache



Security deep-dive

- **Authentication:** JWT, signed/verified by JwtService. JwtAuthFilter runs once per request; if present and valid, SecurityContext is populated.
- **Global roles:** Role.USER (default) and Role.ADMIN. ADMIN can access /api/admin/** and is allowed to operate on any board through explicit controller rules.
- **Per-board roles:** BoardMemberRole = OWNER, MEMBER, VIEWER.
    - Enforced by controller methods (e.g., only OWNER can invite/remove; OWNER+MEMBER can mutate lists/tasks; VIEWER read-only).
    - Helper: BoardSecurity (if present) or service-level checks on BoardMembershipService.
- **Endpoint rules (SecurityConfig):**
    - Permit: /api/auth/signup, /api/auth/login, OPTIONS /**, Swagger (/v3/api-docs/**, /swagger-ui/**).
    - Admin only: /api/admin/**.
    - Everything else: authenticated.
- **CORS:** Origins http://localhost:3000 & http://127.0.0.1:3000, methods GET/POST/PUT/PATCH/DELETE/OPTIONS, credentials allowed.
- **Rate limiting:** RateLimitFilter (in-mem IP counter, 1-min window, 100 req/min). Returns 429 Too Many Requestswhen exceeded.
- **Input hardening:** Bean Validation on DTOs (e.g., @NotBlank, length caps), plus HtmlSanitizer to strip scripts/handlers from free text.
- **Errors:** GlobalExceptionHandler normalizes:
    - 400 for parse/validation/type issues (with top-level message)
    - 403 for access denied/forbidden
    - 404 for missing ids
    - 500 for unhandled exceptions

Data model (ER)



USER		
BIGINT	id	PK
STRING	email	
STRING	name	
STRING	passwordHash	
STRING	role	
DATETIME	createdAt	

BOARD		
BIGINT	id	PK
STRING	name	
BIGINT	ownerId	FK
DATETIME	createdAt	

BOARD_MEMBER		
BIGINT	id	PK
BIGINT	boardId	FK
BIGINT	userId	FK
STRING	role	

BOARD_LIST		
BIGINT	id	PK
BIGINT	boardId	FK
STRING	name	
INT	position	

ACTIVITY_LOG		
BIGINT	id	PK
BIGINT	boardId	FK
BIGINT	actorId	FK
STRING	type	
TEXT	detail	
DATETIME	createdAt	

TASK		
BIGINT	id	PK
BIGINT	listId	FK
STRING	title	
TEXT	description	
DATE	dueDate	
STRING	status	
DATETIME	createdAt	
DATETIME	completedAt	

TASK_ASSIGNEE		
BIGINT	taskId	FK
BIGINT	userId	FK

Caching strategy

- **Cache names:** boardDetails, userProfiles, searchResults (declared in CacheConfig).
- **Hot path:** getBoard(boardId) cached; evicted on:
    - Board rename / delete
    - Membership change
    - Task/list mutations that alter the board aggregate shown by /api/boards/{id}
- **Backend:** ConcurrentMapCacheManager (in-proc).

  Prod option: replace bean with RedisCacheManager (same @Cacheable/@CacheEvict annotations).

Realtime (SSE)

- StreamController exposes /api/boards/{id}/stream: clients open a long-lived SSE connection; on connect, an emitter is registered in SseRegistry.
- TaskService/BoardService call SseRegistry.sendToBoard(boardId, event, payload) after successful mutations.
- **Scaling:** current registry is in-memory; for multi-instance you'll either:
    - a) add sticky sessions at the load balancer for SSE routes, or
    - b) publish change events to Redis (pub/sub), and each instance forwards to its local emitters (recommended).

Email & Scheduler

- **MailService (Spring Mail) sends simple notifications:**
    - On assignment (TaskService.assign) → each assignee
    - On due-soon (DueDateReminderJob) → assignees of tasks due in next 24h
- Dev uses **MailHog** (localhost:1025 SMTP, UI :8025).
- SchedulingConfig enables @Scheduled. Frequency comes from reminders.poll-ms (YAML) / defaults in code.

Analytics (Admins)

- **AdminAnalyticsController (under /api/admin/analytics/**) with:**
    - /board-task-counts → totals of TODO / IN_PROGRESS / DONE per board
    - /avg-completion-per-board → average hours to completion per board (Java post-processing)
    - /most-active-users?limit=N → users with the most ActivityLog rows
- Guarded by @PreAuthorize("hasRole('ADMIN')").
- Queries are JPQL and only read existing entities; no extra schema needed.

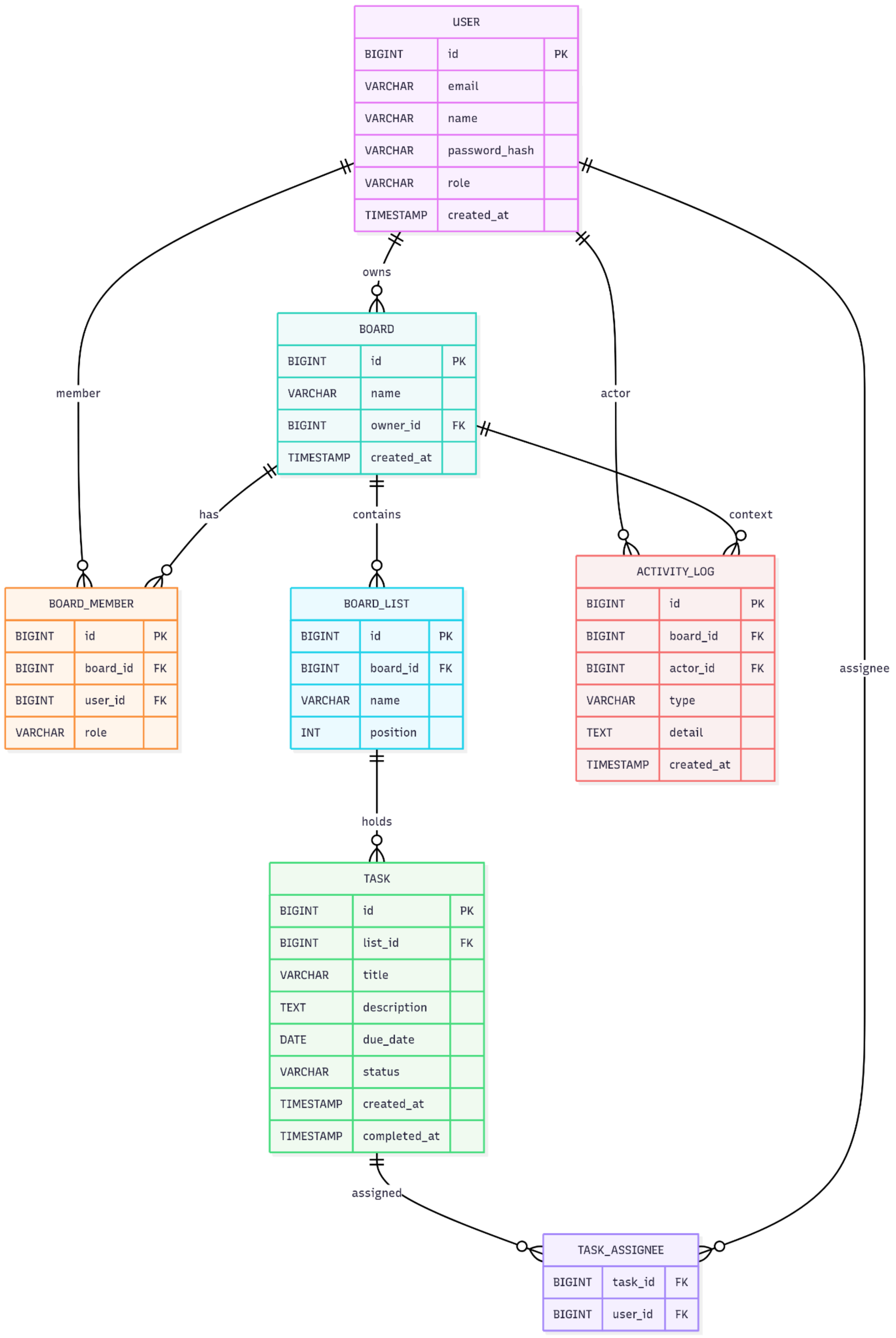
Observability & ops

- **Logs:** logging.level.com.example.taskmanager=DEBUG, errors centralized by GlobalExceptionHandler.
- **Actuator:** health/info enabled (extend as needed).
- **Swagger:** /swagger-ui & /v3/api-docs.
- **Profiles:** dev allows H2 console (/h2-console), relaxed headers for frames; stateless otherwise.

Scaling & reliability notes

- **Stateless app**: JWT + no HTTP session → easy horizontal scaling.
- **SSE**: use Redis pub/sub or LB stickiness (see §9).
- **Rate limiting**: in-memory; for multi-node move to a distributed limiter (e.g., bucket4j + Redis).
- **Idempotency**: consider idempotency keys on mutating endpoints if clients may retry.
- **Consistency**: side-effects are intentionally *best-effort*—they won't roll back the main transaction. If you need guaranteed delivery, enqueue to a message broker (Kafka/SQS/Rabbit) and process asynchronously.
- **Indexes**: add DB indexes on task(list_id), task(status), task(dueDate), task_assignee(task_id,user_id), board_member(board_id,user_id).

ER Diagram;



USER
BIGINT id PK
VARCHAR email
VARCHAR name
VARCHAR password_hash
VARCHAR role
TIMESTAMP created_at
owns
member
actor
assignee
BOARD
BIGINT id PK
VARCHAR name
BIGINT owner_id FK
TIMESTAMP created_at
has
contains
context
BOARD_MEMBER
BIGINT id PK
BIGINT board_id FK
BIGINT user_id FK
VARCHAR role
BOARD_LIST
BIGINT id PK
BIGINT board_id FK
VARCHAR name
INT position
ACTIVITY_LOG
BIGINT id PK
BIGINT board_id FK
BIGINT actor_id FK
VARCHAR type
TEXT detail
TIMESTAMP created_at
holds
TASK
BIGINT id PK
BIGINT list_id FK
VARCHAR title
TEXT description
DATE due_date
VARCHAR status
TIMESTAMP created_at
TIMESTAMP completed_at
assigned
TASK_ASSIGNEE
BIGINT task_id FK
BIGINT user_id FK

What each table stores

- **USER**
    - role: USER or ADMIN (enforce with a CHECK).
    - Unique: email.
- **BOARD**
    - owner_id → USER.id (FK).
- **BOARD_MEMBER**
    - role: OWNER, MEMBER, VIEWER.
    - Uniqueness: (board_id, user_id).
    - FKs: board_id → BOARD.id, user_id → USER.id.
- **BOARD_LIST**
    - FKs: board_id → BOARD.id.
    - position is the order within a board.
    - Optional unique: (board_id, position).
- **TASK**
    - FKs: list_id → BOARD_LIST.id.
    - status: TODO, IN_PROGRESS, DONE.
    - Optional index: (list_id, status) and (due_date).
- **TASK_ASSIGNEE**
    - Join table for many-to-many tasks↔users.
    - PK: (task_id, user_id) and FKs to TASK.id and USER.id.
- **ACTIVITY_LOG**
    - FKs: board_id → BOARD.id, actor_id → USER.id.
    - type examples: BOARD_UPDATED, BOARD_MEMBER_ADDED, TASK_CREATED, TASK_UPDATED, TASK_DELETED.

Postgres DDL sketch (matches the ERD)

```
CREATE TABLE users (
  id BIGSERIAL PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE,
  name VARCHAR(255) NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role VARCHAR(16) NOT NULL CHECK (role IN ('USER','ADMIN')),
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE TABLE boards (
  id BIGSERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  owner_id BIGINT NOT NULL REFERENCES users(id) ON DELETE RESTRICT,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE TABLE board_members (
  id BIGSERIAL PRIMARY KEY,
  board_id BIGINT NOT NULL REFERENCES boards(id) ON DELETE CASCADE,
  user_id  BIGINT NOT NULL REFERENCES users(id)  ON DELETE CASCADE,
  role VARCHAR(16) NOT NULL CHECK (role IN ('OWNER','MEMBER','VIEWER')),
  UNIQUE (board_id, user_id)
);

CREATE TABLE board_lists (
  id BIGSERIAL PRIMARY KEY,
  board_id BIGINT NOT NULL REFERENCES boards(id) ON DELETE CASCADE,
  name VARCHAR(255) NOT NULL,
  position INT NOT NULL,
  UNIQUE (board_id, position)
);

CREATE TABLE tasks (
  id BIGSERIAL PRIMARY KEY,
  list_id BIGINT NOT NULL REFERENCES board_lists(id) ON DELETE CASCADE,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  due_date DATE,
  status VARCHAR(16) NOT NULL CHECK (status IN ('TODO','IN_PROGRESS','DONE')),
  created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  completed_at TIMESTAMPTZ
);

CREATE INDEX idx_tasks_list_status ON tasks(list_id, status);
CREATE INDEX idx_tasks_due_date   ON tasks(due_date);

CREATE TABLE task_assignees (
  task_id BIGINT NOT NULL REFERENCES tasks(id) ON DELETE CASCADE,
  user_id BIGINT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  PRIMARY KEY (task_id, user_id)
);

CREATE TABLE activity_logs (
  id BIGSERIAL PRIMARY KEY,
  board_id BIGINT NOT NULL REFERENCES boards(id) ON DELETE CASCADE,
  actor_id BIGINT NOT NULL REFERENCES users(id)  ON DELETE SET NULL,
  type VARCHAR(64) NOT NULL,
  detail TEXT,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Optional performance helpers
CREATE INDEX idx_activity_board_time ON activity_logs(board_id, created_at DESC);
CREATE INDEX idx_board_members_user  ON board_members(user_id);
```

API endpoint list with request/response examples

Auth API

Base

/api/auth

Sign up

POST /api/auth/signup

Body (JSON) — from SignupRequest

```
{
 "email": "alice@example.com",
 "password": "secret123",
 "name": "Alice"
}
```

Validations

- email: required, valid email
- password: required, 6–100 chars
- name: optional, ≤100 chars

Response 200 — AuthResponse (token included)

```
{
 "userId": 1,
 "email": "alice@example.com",
 "name": "Alice",
 "role": "USER",
 "token": "<jwt>"
}
```

Errors

- 400 Bad Request
  - {"status":400,"error":"Bad Request","message":"Email already registered","path":"/api/auth/signup"}
  - Or validation messages like password: size must be between 6 and 100
- 500 on unexpected exceptions

Login

POST /api/auth/login

Body (JSON) — from LoginRequest

```
{
 "email": "alice@example.com",
 "password": "secret123"
}
```

Response 200 — AuthResponse (token included)

```
{
 "userId": 1,
 "email": "alice@example.com",
 "name": "Alice",
 "role": "USER",
 "token": "<jwt>"
}
```

Errors

- 400 Bad Request for bad payload / type mismatch
- 401 Unauthorized for bad credentials (raised by Spring Security auth manager; returned by entry point as plain 401 with no JSON body in this config)

Me (profile)

GET /api/auth/me

Auth required: Authorization: Bearer <jwt>

Response 200 — AuthResponse (token **null** by design)

```
{
  "userId": 1,
  "email": "alice@example.com",
  "name": "Alice",
  "role": "USER",
  "token": null
}
```

Errors

- 401 Unauthorized if no/invalid JWT (controller throws ResponseStatusException(401))

Errors

- 401 Unauthorized if no/invalid JWT (controller throws ResponseStatusException(401))

AuthN / AuthZ Rules (from SecurityConfig, JwtAuthFilter, UserDetailsServiceImpl)

- **Public endpoints**
  - POST /api/auth/signup
  - POST /api/auth/login
  - Swagger/OpenAPI: /swagger-ui/**, /v3/api-docs/**
  - OPTIONS /** (CORS preflight)
  - Actuator: /actuator/health, /actuator/info
- **Admin-only**: anything under /api/admin/** requires role ADMIN.
- **All other paths**: require a valid **JWT**.

JWT details (from JwtService)

- Algorithm: HS256
- Claims: sub = email, role = "ROLE_USER" or "ROLE_ADMIN"
- Exp: 24h
- Header to send:
  Authorization: Bearer <token>
- The filter (JwtAuthFilter) populates Spring Security Authentication if token is valid; otherwise request proceeds unauthenticated and hits the 401/403 handlers.

CORS

Allowed origins: http://localhost:3000, http://127.0.0.1:3000

Methods: GET, POST, PUT, PATCH, DELETE, OPTIONS

Headers: *, credentials allowed.

Error Envelope (from GlobalExceptionHandler)

When exceptions are handled by the advice, JSON responses look like:

```
{
  "status": 400,
  "error": "Bad Request",
  "message": "field: message; field2: message2",
  "path": "/api/..."
}
```

Mapped cases

- **403 Forbidden**
  - Custom ForbiddenException → {"status":403,"error":"Forbidden","message":"<reason>","path":"..."}
  - Spring AccessDeniedException → message "Access is denied"
- **404 Not Found**
  - NotFoundException
- **400 Bad Request**
  - MethodArgumentNotValidException (bean validation): concatenated field: message
  - ConstraintViolationException, HttpMessageNotReadableException, MethodArgumentTypeMismatchException, IllegalArgumentException, DataIntegrityViolationException, TransactionSystemException → first-line message
- **500 Internal Server Error**
  - Any other exception → message "Unexpected error"

Note: Authentication failures (no/invalid token, bad credentials) are returned by the security entry point as **401** with an empty body (not the error envelope), per your exceptionHandling configuration.

Boards, Lists & Membership API

Auth: All endpoints below require Authorization: Bearer <jwt> unless noted otherwise.
RBAC checks: Done with @PreAuthorize against @boardSecurity:

- isMember(boardId, auth) → any member (OWNER/MEMBER/VIEWER)
- canEdit(boardId, auth) → OWNER or MEMBER
- isOwner(boardId, auth) → OWNER only

Boards

List my boards

GET /api/boards

Returns boards where the current user is a member.

200 Response — BoardDto[]

```
[
  {
    "id": 12,
    "name": "Engineering",
    "lists": [
      { "id": 101, "name": "To Do", "position": 1 },
      { "id": 102, "name": "In Progress", "position": 2 }
    ]
  }
]
```

Get board (with lists & tasks summary)

GET /api/boards/{id}

- **RBAC**: member of the board (else **403**).

200 Response — BoardWithListsDto

```
{

 "id": 12,

 "name": "Engineering",

 "ownerId": 7,

 "lists": [

   {

     "id": 101,

     "name": "To Do",

     "position": 1,

     "tasks": [

       { "id": 9001, "title": "Set up CI", "status": "TODO" },

       { "id": 9002, "title": "Write tests", "status": "IN_PROGRESS" }

     ]

   }

 ]
```

}

Errors

- 403 Forbidden if not a member.
- 404 Not Found if board id doesn't exist.

Create board

POST /api/boards

Body — CreateBoardRequest

```
{ "name": "Marketing" }
```

201/200 Response — BoardDto

```
{
  "id": 42,
  "name": "Marketing",
  "lists": []
}
```

Notes:

- Service sets current user as owner and also inserts a BoardMember with role OWNER.

Rename board (owner only)

PUT /api/boards/{id}

Body — same shape as CreateBoardRequest

```
{ "name": "Marketing Q4" }
```

200 Response

```
{ "id": 42, "name": "Marketing Q4", "lists": [] }
```

Errors

- 400 Bad Request if name missing/blank.
- 403 Forbidden/401 depending on auth.
- 404 Not Found if board doesn't exist.

Board Memberships

Base: /api/boards/{boardId}/members

Controller currently returns the BoardMember entity directly. Because board and user fields are @JsonIgnore, responses expose only:

```
{ "id": 123, "role": "MEMBER" }
```

If you want rich member details (email, name, role), consider returning BoardMemberResponse instead.

List members

GET /api/boards/{boardId}/members

RBAC: member

200 Response

```
[
  { "id": 555, "role": "OWNER" },
  { "id": 556, "role": "MEMBER" }
]
```

Invite / add member

POST /api/boards/{boardId}/members

RBAC: owner

Query params

- userId (Long, required)
- role (enum OWNER|MEMBER|VIEWER, required)

Example

```
POST /api/boards/42/members?userId=7&role=MEMBER
```

200 Response

```
{ "id": 556, "role": "MEMBER" }
```

Errors

- 403 if caller is not owner
- 404 if board or user not found

Update member role

PATCH /api/boards/{boardId}/members/{userId}

RBAC: owner

Query params

- role (enum OWNER|MEMBER|VIEWER, required)

Example

```
PATCH /api/boards/42/members/7?role=VIEWER
```

200 Response

```
{ "id": 556, "role": "VIEWER" }
```

Errors

- 404 if membership not found

Remove member

DELETE /api/boards/{boardId}/members/{userId}

RBAC: owner

204 No Content on success.

Lists (Kanban columns)

```
Base: /api/boards/{boardId}/lists
```

> Responses return the **BoardList entity**. Because board and tasks are @JsonIgnore, JSON shows only: id, name, position.

List lists in a board

GET /api/boards/{boardId}/lists

RBAC: member

200 Response

```
[
  { "id": 101, "name": "To Do", "position": 1 },
  { "id": 102, "name": "In Progress", "position": 2 }
]
```

Create a list

POST /api/boards/{boardId}/lists

RBAC: canEdit (owner or member)

Body — CreateListRequest

```
{ "name": "Done", "position": 3 }
```

200 Response

```
{ "id": 103, "name": "Done", "position": 3 }
```

Errors

- 403 if caller is viewer / not member
- 404 if board not found
- 400 if name blank (bean validation)

Update a list

PUT /api/boards/{boardId}/lists/{listId}

RBAC: canEdit

Body — UpdateListRequest (all optional)

```
{ "name": "Backlog", "position": 0 }
```

200 Response

```
{ "id": 101, "name": "Backlog", "position": 0 }
```

Errors

- 404 if list not found
- 403 if not allowed

Delete a list

DELETE /api/boards/{boardId}/lists/{listId}

RBAC: canEdit

204 No Content on success.

Errors

- 404 if list not found
- 403 if not allowed

Notes on serialization

- BoardDto & BoardWithListsDto are safe DTOs for read models.
- BoardMember and BoardList use @JsonIgnore to avoid lazy-loading issues; that intentionally limits the JSON fields. If you want richer API responses, return DTOs (e.g., BoardMemberResponse) from controllers.

Task API

Auth: All endpoints require Authorization: Bearer <jwt>.
Notes on JSON: Your controllers return **entity objects** (Task) for create/update/read. Because Task.assignees and Task.list are annotated @JsonIgnore, responses won't include those fields unless you switch to DTOs later.

Lists → Tasks

List tasks in a list

GET /api/lists/{listId}/tasks

200 Response — Task[]

```
[
  {
    "id": 9001,
    "title": "Set up CI",
    "description": "Use GitHub Actions",
    "dueDate": "2025-09-01",
    "status": "TODO",
    "createdAt": "2025-08-31T18:42:10.123Z",
    "completedAt": null
  }
]
```

Errors: 404 List not found.

Create task in a list

POST /api/lists/{listId}/tasks

Body — CreateTaskRequest

```
{
  "title": "Write tests",
  "description": "Cover service layer",
  "dueDate": "2025-09-05"
}
```

200 Response — Task

```
{
  "id": 9002,
  "title": "Write tests",
  "description": "Cover service layer",
  "dueDate": "2025-09-05",
  "status": "TODO",
  "createdAt": "2025-08-31T18:48:11.010Z",

  "completedAt": null

}
```

Validations: title required, ≤255; description ≤5000.

Side effects: Activity log TASK_CREATED; SSE "task-changed"; email later when assignees are set.

Update task (PUT – full-ish update)

PUT /api/lists/{listId}/tasks/{taskId}

Body — UpdateTaskRequest (all fields optional; omitted fields stay unchanged)

```
{

  "title": "Write integration tests",

  "description": "Spring @WebMvcTest",

  "dueDate": "2025-09-06",

  "status": "IN_PROGRESS"

}
```

200 Response — updated Task.

Behavior:

- If status becomes DONE, completedAt is set to now.
- If status changes from DONE to something else, completedAt is cleared.
- Activity log TASK_UPDATED; SSE "task-changed".

Errors: 404 Task not found.

Delete task

DELETE /api/lists/{listId}/tasks/{taskId}

204 No Content on success.

Behavior: Activity log TASK_DELETED; SSE "task-changed".

Errors: 404 Task not found.

Task-scoped (without list in path)

Patch task (partial update)

PATCH /api/tasks/{taskId}

Body — UpdateTaskRequest (all optional)

```
{ "status": "DONE" }
```

200 Response — updated Task (with completedAt set).

Errors: 404 Task not found.

Assignees

You have two equivalent entry points that call the same service:

- **PUT** /api/lists/{listId}/tasks/{taskId}/assignees
- **PUT** /api/tasks/{taskId}/assignees

Body — AssignUsersRequest / AssignRequest

```
{ "userIds": [7, 8] }
```

200 Response — updated Task (note: assignees are @JsonIgnore, so they won't appear in JSON).

Behavior: Replaces the entire set of assignees; sends an email per assignee via MailService; logs activity and emits SSE "TASK_ASSIGNED".

Errors: 404 Task not found.

Search

Two routes (same behavior); choose whichever you prefer:

- **GET** /api/tasks/search?boardId={boardId}&q=...&status=...&assigneeId=...&from=YYYY-MM-DD&to=YYYY-MM-DD&page=0&size=20
- **GET** /api/boards/{boardId}/tasks/search?q=...&status=...&assigneeId=...&from=...&to=...&page=...&size=...

RBAC: @boardSecurity.isMember(boardId, authentication).

200 Response — Spring Data Page<TaskSummaryDto>

```
{
 "content": [
  {
   "id": 9002,
   "listId": 101,
   "boardId": 12,
   "title": "Write tests",
   "description": "Cover service layer",
   "dueDate": "2025-09-05",
   "status": "IN_PROGRESS",
   "createdAt": "2025-08-31T18:48:11.010Z",

   "completedAt": null

  }

 ],

 "pageable": { "sort": { "sorted": true, "unsorted": false, "empty": false }, "pageNumber": 0, "pageSize": 20, "offset": 0, "paged": true,
"unpaged": false },

 "totalPages": 1,

 "totalElements": 1,

 "last": true,

 "size": 20,

 "number": 0,

 "sort": { "sorted": true, "unsorted": false, "empty": false },

 "first": true,

 "numberOfElements": 1,

 "empty": false

}
```

Sorting: by createdAt DESC (hard-coded).

Filters supported:

- q → title or description contains (case-insensitive)
- status ∈ TODO|IN_PROGRESS|DONE
- assigneeId → tasks assigned to user
- from, to → due date range

Activity & Realtime API

Auth: Authorization: Bearer <jwt> required for all endpoints below.
RBAC: Membership is enforced with @boardSecurity.isMember(boardId, authentication) unless noted as ADMIN.

Activity Feed

List board activity (paginated)

GET /api/boards/{boardId}/activity?page=0&size=20

- **RBAC:** board **member** (or ADMIN)
- **Paging:** Spring Data Page<ActivityLog>; size clamped to 1…100 (default 20), page ≥ 0
- **Sort:** newest first (createdAt DESC)

200 Response

```
{
  "content": [
    {
      "id": 12345,
      "type": "TASK_CREATED",
      "detail": "Task \"Write tests\" created",
      "createdAt": "2025-08-31T19:04:13.582Z"
    }
  ],
  "pageable": { /* Spring Data metadata */ },
  "totalPages": 3,
  "totalElements": 47,
  "first": false,
  "last": false,
  "size": 20,
  "number": 1,
  "numberOfElements": 20,
  "sort": { "sorted": true, "unsorted": false, "empty": false },
  "empty": false
}
```

Internals (not serialized): each row is linked to a board and optional actor via @JsonIgnore.
Indexes on (board_id, createdAt) keep this query fast.

Possible type values (from ActivityType enum):

BOARD_MEMBER_ADDED, BOARD_MEMBER_REMOVED, BOARD_UPDATED,

LIST_CREATED, LIST_UPDATED, LIST_DELETED,

TASK_CREATED, TASK_UPDATED, TASK_MOVED, TASK_DELETED,

USER_JOINED_BOARD

Errors

- 404 Board not found (service checks board existence before querying)
- 403 Forbidden if not a member

Purge board activity (admin only)

DELETE /api/boards/{boardId}/activity

- RBAC: hasRole('ADMIN')
- 204 No Content on success
- Errors: 404 Board not found (service validates board first)

Realtime: Server-Sent Events (SSE)

Subscribe to board stream

GET /api/boards/{boardId}/stream

Headers: Accept: text/event-stream

RBAC: board **member**

- Keeps a live connection open for ~30 minutes (SseRegistry.DEFAULT_TIMEOUT_MS); client should auto-reconnect on close.
- On connect, server sends an initial connected event.

Example event stream

```
event: connected

id: 1693500000000

data: {"ts":"2025-08-31T19:10:22.001Z","boardId":12}


event: task-changed

id: 1693500001573

data: {"type":"TASK_CREATED","taskId":9002}


event: task-changed

id: 1693500004123

data: {"type":"TASK_UPDATED","taskId":9002}


event: task-changed

id: 1693500010555

data: {"type":"TASK_ASSIGNED","taskId":9002}
```

Where do these come from?

- TaskService.create → "task-changed" with {type:"TASK_CREATED", taskId}

- TaskService.update/changeStatus → "task-changed" with {type:"TASK_UPDATED", taskId}
- TaskService.delete → "task-changed" with {type:"TASK_DELETED", taskId}
- TaskService.assign → "task-changed" with {type:"TASK_ASSIGNED", taskId}

Notifications & Scheduler

Overview

- Outbound email is sent via a thin facade: MailService.
- No public endpoints; emails are triggered by domain actions and a scheduled job.
- Works even without SMTP configured (logs the email instead), which is great for dev.

When emails are sent

Task assignment

Triggered by: TaskService.assign(...)

Template (subject/body built in MailService.sendTaskAssigned)

Subject: [Task Assigned] <taskTitle>

Body: You were assigned to task: <taskTitle> (Board: <boardName>).

Recipients: all users in the new assignee set

Side effects: independent of activity feed/SSE (those are also emitted by assign())

Due-date reminders (next 24 hours)

Component: DueDateReminderJob (Spring @Scheduled)

Query window: LocalDate.now() .. +1 day (inclusive)

Eligibility:

- Task.status != DONE
- Task.dueDate between **today** and **tomorrow**
- User must be an assignee

De-dupe: 1 email per (assigneeEmail, taskId) per run

Email template (from sendDueSoonReminder)

Subject: [Task Due Soon] <taskTitle>

Body:

Reminder: "<taskTitle>" is due on <dueDate> (Board: <boardName>).

Please review and complete it.

Run output: logs DueDateReminderJob sent N reminder email(s).

API Reference (derived from code)

Conventions

- **Base URL:** http://localhost:8080
- **Auth:** JWT in Authorization: Bearer <token>
- **Content-Type:** application/json
- **Time zone:** UTC (per application.yml)
- **Errors:** Unified JSON from GlobalExceptionHandler

```
{ "status":400, "error":"Bad Request", "message":"...", "path":"/api/..." }
```

- **Security defaults:**
  - /api/auth/signup, /api/auth/login, /swagger-ui/**, /v3/api-docs/** are public
  - /api/admin/** requires ROLE_ADMIN
  - everything else requires a valid JWT
  - 401 for unauthenticated, 403 for unauthorized (Spring Security + @PreAuthorize)

Auth

POST

/api/auth/signup

Body (SignupRequest):

```
{ "email":"alice@example.com", "password":"secret123", "name":"Alice" }
```

Response (AuthResponse 200):

```
{

  "userId": 1, "email": "alice@example.com", "name": "Alice",

  "role": "USER", "token": "eyJhbGciOiJIUzI1NiJ9..."

}
```

Notes: 400 if email exists; passwords are BCrypt'd. Role defaults to USER.

POST

/api/auth/login

Body (LoginRequest):

```
{ "email": "alice@example.com", "password": "secret123" }
```

Response (AuthResponse 200): same shape as signup.

GET

/api/auth/me

Headers: Authorization: Bearer ...

Response (AuthResponse without token):

```
{ "userId":1, "email":"alice@example.com", "name":"Alice", "role":"USER", "token": null }
```

Errors: 401 if no/invalid JWT.

Boards

GET

/api/boards

Returns boards where current user is a member.

Response (BoardDto[]):

```
[
  { "id":1, "name":"Demo Board", "lists":[ /* see ListDto */ ] }
]
```

GET

/api/boards/{id}

Authz: member of board (checked via BoardSecurity.isMember)

Response (BoardWithListsDto):

```
{
  "id":1, "name":"Demo Board (renamed)", "ownerId":1,
  "lists":[
    {
      "id": 10, "name":"To Do", "position":1,
      "tasks":[ { "id":100, "title":"Write tests", "status":"DONE" } ]
    }
  ]
}
```

Errors: 403 if not a member, 404 if board not found.

POST

/api/boards

Body (CreateBoardRequest):

```
{ "name":"New Board" }
```

Response (BoardDto 200).

PUT

/api/boards/{id}

Owner-only rename (owner determined by Board.owner).

Body (CreateBoardRequest / UpdateBoardRequest):

```
{ "name":"Renamed Board" }
```

Response (BoardDto).

Errors: 400 if blank name, 403 if not owner, 404 if board not found.

DELETE

/api/boards/{id}

Owner-only (enforced in BoardService.delete() via BoardSecurity.isOwner).

Side-effects: deletes activity logs for board, clears task assignees, deletes tasks, lists, memberships, then board.

Response: 200 OK with empty body.

Board Memberships

Base path: /api/boards/{boardId}/members

GET

Authz: member (@boardSecurity.isMember)

Response (array of BoardMember):

```
[
  { "id":1, "role":"OWNER" },
  { "id":2, "role":"MEMBER" },
  { "id":3, "role":"VIEWER" }
]
```

(Note: board and user fields are @JsonIgnore.)

POST

Owner-only. Invite or upsert role.

Query params: ?userId=<id>&role=OWNER|MEMBER|VIEWER

Response (BoardMember):

```
{ "id": 5, "role": "MEMBER" }
```

PATCH

/{userId}

Owner-only. Update role.

Query param: ?role=OWNER|MEMBER|VIEWER

Response (BoardMember).

DELETE

/{userId}

Owner-only. Remove member.

Response: 200 OK empty.

Lists

Base path: /api/boards/{boardId}/lists

GET

Authz: member

Response (BoardList[]—entity with tasks ignored in JSON):

```
[
  { "id":1, "name":"To Do", "position":1 },
  { "id":2, "name":"In Progress", "position":2 }
]
```

POST

Authz: can edit (owner or member; boardSecurity.canEdit)

Body (CreateListRequest):

```
{ "name": "To Do", "position": 1 }
```

Response (BoardList).

PUT

/{listId}

Authz: can edit

Body (UpdateListRequest—both fields optional):

```
{ "name":"Doing", "position":2 }Response (BoardList).
```

DELETE

/{listId}

Authz: can edit

Response: 200 OK empty.

Tasks

List-scoped endpoints

Base: /api/lists/{listId}/tasks

GET

Returns tasks in list (membership is indirectly validated via list existence; optional to add BoardSecurity check).

Response (Task):

```
[
  {
    "id": 100,
    "title": "Write tests",
    "description": "updated",
    "dueDate": "2030-01-02",
    "status": "DONE",
    "createdAt": "2025-08-31T02:14:31.013523Z",
    "completedAt": "2025-08-31T02:14:31.052588Z"
  }
]
```

(Fields list and assignees are @JsonIgnore.)

POST

Body (CreateTaskRequest):

```
{ "title":"Write tests", "description":"Stage 1 check", "dueDate":"2030-01-01" }
```

Response (Task).

Sanitization: title/description passed through HtmlSanitizer (strips <script>, on*= attrs, javascript:).

Validation: title @NotBlank ≤ 255, description ≤ 5000.

PUT

/{taskId}

Full update (idempotent-ish).

Body (UpdateTaskRequest):

```
{
  "title":"Write tests (upd)",
  "description":"updated",
  "dueDate":"2030-01-02",
  "status":"IN_PROGRESS"
}
```

Response (Task).

If status becomes DONE, completedAt is set to now. If changed away from DONE, completedAt cleared.

DELETE

/{taskId}

Response: 200 OK empty.

Task-scoped endpoints

PATCH

/api/tasks/{taskId}

Partial update; fields optional (same body shape as UpdateTaskRequest).

Response (Task).

PUT

/api/tasks/{taskId}/assignees

Replace assignees.

Body:

```
{ "userIds": [1,2,3] }Response (Task).
```

Side-effects: sends an assignment email per user; emits SSE (task-changed with {type:"TASK_ASSIGNED", taskId}); activity log.

Status shortcut

PUT

/api/lists/{listId}/tasks/{taskId}/status

Body (UpdateTaskStatusRequest):

```
{ "status": "DONE" }
```

Response (Task).

Task Search

Two equivalent routes (both require membership):

GET

/api/tasks/search

GET

/api/boards/{boardId}/tasks/search

Query params:

- boardId (required on the first route; path variable on the second)
- q (title/description contains, case-insensitive)
- status (TODO|IN_PROGRESS|DONE)
- assigneeId (user id)
- from / to (ISO YYYY-MM-DD due-date range)
- page (default 0), size (default 20, max 100)

Response (Page<TaskSummaryDto>):

```
{

  "content": [
    {
      "id": 2,
      "listId": 1,
      "boardId": 1,
      "title": "Write tests",
      "description": "stage2 tests",
      "dueDate": "2030-01-02",
      "status": "DONE",
      "createdAt": "2025-08-31T02:16:00.232616Z",
      "completedAt": "2025-08-31T02:16:00.252275Z"
    }
  ],
  "number": 0, "size": 20, "totalElements": 1, "totalPages": 1,
  "first": true, "last": true, "sort": { "sorted": true, "unsorted": false, "empty": false },
  "pageable": { "pageNumber":0, "pageSize":20, "paged":true, ... }
}
```

Activity Feed

GET

/api/boards/{boardId}/activity

Authz: member

Query: page (0), size (20; clamp 1..100)

Response (Page<ActivityLog>):

```
{
  "content":[
    { "id": 7, "type":"TASK_UPDATED", "detail":"Task \"Write tests\" updated", "createdAt":"2025-08-31T02:16:00.252976Z" }
  ],
  "number":0, "size":20, "totalElements":8, "totalPages":1, ...
}
```

(Fields board and actor are @JsonIgnore in entity JSON.)

DELETE

/api/boards/{boardId}/activity

Admin-only purge. Response: 200 OK empty.

Real-Time (SSE)

GET

/api/boards/{boardId}/stream

Produces text/event-stream. Authz: member.

Immediately emits a connected event, then domain events:

```
event: connected

data: {"ts":"2025-08-31T02:17:42.123136Z","boardId":1}

id: 1756606662123

retry: 2000
```

Server also emits task-changed with small JSON payloads, e.g.:

```
{ "type":"TASK_CREATED", "taskId": 123 }
```

Admin Analytics (ADMIN only)

Base: /api/admin/analytics

GET

/board-task-counts

Totals by board. Response:

```
[
  { "boardId":1, "boardName":"Stage4 Board cache test", "todo":3, "inProgress":0, "done":1 }
]
```

GET

/avg-completion-per-board

Average hours to complete (created→completed), Java-computed. Response:

```
[
```

{ "boardId":1, "boardName":"Stage4 Board cache test", "avgHours": 0.026944444444444444 }

]

GET

/most-active-users?limit=5

Top N users by number of ActivityLog entries (actor present). Response:

[

{ "userId":1, "email":"alice@example.com", "activityCount":3 }

]

Security, Validation, Rate Limiting

- **Role model (global):**
  - JWT contains "role": "ROLE_USER|ROLE_ADMIN"; resolved by JwtAuthFilter + UserDetailsServiceImpl.
  - @PreAuthorize checks on controllers, and explicit service checks via BoardSecurity methods.
- **Board roles (membership):** OWNER, MEMBER, VIEWER (enum BoardMemberRole)
  - *Owner* can manage board settings & memberships
  - *Member* can modify lists/tasks
  - *Viewer* read-only
- **Validation:** request DTO constraints (e.g., @NotBlank, @Size). On violation: 400 with combined field messages.
- **Sanitization:** HtmlSanitizer.sanitize on task titles/descriptions removes scripts/event handlers/javascript: URLs and trims overly long inputs.
- **Rate limiting:** naive IP counter (RateLimitFilter)
  - Defaults: 1-minute window, 100 requests.
  - Env overrides: RATELIMIT_WINDOW_MS, RATELIMIT_MAX.
  - Over limit → 429 Too Many Requests (+ Retry-After header).

How authentication and role-based access are handled

Authentication (JWT, Stateless)

- **Credentials flow**
  - POST /api/auth/signup → creates a User with Role.USER, hashes password with **BCrypt**, issues a **JWT**.
  - POST /api/auth/login → authenticates via AuthenticationManager + DaoAuthenticationProvider, issues a **JWT**.
- **JWT details**
  - Implemented in JwtService using **HS256** (jjwt).
  - Claims:
    - sub = user email
    - role = "ROLE_USER" or "ROLE_ADMIN"

        - exp = now + 24h
    - Token returned only on signup/login. GET /api/auth/me returns profile **without** token.
- **Password storage**
    - BCrypt via BCryptPasswordEncoder (strong, salted, adaptive).
- **Request processing**
    - Client sends Authorization: Bearer <token>.
    - JwtAuthFilter runs before username/password auth.
        - Skips public paths (/api/auth/signup, /api/auth/login, Swagger, health, etc.).
        - Extracts/validates token via JwtService.
        - Loads user with UserDetailsServiceImpl, sets SecurityContext with ROLE_USER or ROLE_ADMIN.
    - Sessions are **stateless** (SessionCreationPolicy.STATELESS).
- **Unauthorized handling**
    - Invalid/missing token → 401 Unauthorized.
    - Error shape provided by GlobalExceptionHandler.
- **CORS / H2 console**
    - CORS allows localhost frontends.
    - /h2-console/** is permitted in dev profile.

Global Role Model (Application Roles)

- Role enum: USER (default) and ADMIN.
- Spring authorities are "ROLE_USER" or "ROLE_ADMIN".
- SecurityConfig guards:
    - Public: /api/auth/signup, /api/auth/login, /swagger-ui/**, /v3/api-docs/**.
    - **Admin-only:** /api/admin/** → requires ROLE_ADMIN.
    - All other endpoints require authentication.

Board-Level Authorization (Resource Roles)

- Each board uses **membership roles** (BoardMemberRole):
    - OWNER – full access (board settings, memberships).
    - MEMBER – can modify lists/tasks.
    - VIEWER – read-only.
- **Enforcement**
    - **Controller annotations** with @PreAuthorize and SpEL:
        - @boardSecurity.isMember(#boardId, authentication)
        - @boardSecurity.canEdit(#boardId, authentication)
        - @boardSecurity.isOwner(#boardId, authentication)
    - **Service checks** also verify ownership or membership before mutating.
- **Typical rules**
    - **Read board/lists/activity/SSE:** any member (OWNER/MEMBER/VIEWER).
    - **Create/update/delete lists & tasks:** OWNER or MEMBER.
    - **Rename/delete board & manage members:** OWNER only.
    - **Admin analytics:** requires ROLE_ADMIN.

Principal Resolution

- Controllers use either:
    - Principal principal (email from JWT subject → DB lookup), or
    - AuthUtils.currentUserId(Authentication) helper.

Input Validation & Sanitization

- Request DTOs use jakarta.validation (@NotBlank, @Size, etc.).
- Violations return 400 Bad Request with field error messages.
- Free-text fields sanitized by HtmlSanitizer: removes <script>, javascript: URLs, and inline event handlers.

Rate Limiting

- RateLimitFilter applies simple IP-based throttling.
    - Default: **100 requests/min** (configurable with RATELIMIT_WINDOW_MS, RATELIMIT_MAX).
    - Exceeding → 429 Too Many Requests + Retry-After.

SSE & Email Security

- **SSE stream** (GET /api/boards/{boardId}/stream) requires board membership.
- **Email notifications** (assignment & due-soon reminders) are triggered by backend services; failures are logged, not surfaced to clients.

Admin vs Member Precedence

- **ROLE_ADMIN** can call /api/admin/** endpoints globally.
- Board resources themselves still check board membership; ADMIN is not automatically a board OWNER.

Security Hardening Notes

- Move JWT secret out of code → environment/config, rotate regularly.
- JWT TTL is 24h → consider refresh tokens for production.
- Prefer DTOs over entities for public API responses to avoid leaking internals.