Give me a seperate list of actionables (+ features) that needs to be done in the task management project that we are building.

Yes. Treat this as a concrete backlog you can tick off. The goal is to have one serious, interview-ready Task Management API showing multi-tenant, production-style backend skills.[1] [2]

Core Domain & Multi-Tenancy

- Define domain entities: `Tenant`, `User`, `Project`, `Board`, `Task`, `Comment`, `Label` with relationships (Tenant → Projects, Project → Boards, Board → Tasks, Task → Comments, many-to-many Task ↔ Label).[1]
- Add multi-tenant support:
  - Every row (Project/Board/Task/Comment) includes `TenantId`.[3] [4]
  - Middleware to resolve current tenant from header / subdomain and scope all queries by `TenantId`.[5] [6]

API & CRUD Features

- Implement CRUD endpoints (DTOs + controllers) for:
  - Tenants (admin only), Projects, Boards, Tasks, Comments, Labels.
- Support search/filter/sort/pagination:
  - Filter tasks by status, assignee, due date range, labels.
  - Sort tasks (due date, created date, priority).
  - Paginate all list endpoints (page, pageSize).[1]

Auth, Users, and RBAC

- JWT authentication:
  - Register/login endpoints, password hashing, token issuing.[7] [8]
- Authorization:
  - Roles: `TenantAdmin`, `Member`. Only admins can manage tenants/projects; members manage tasks.[9] [10]

- Policy-based authorization attributes on controllers/actions.

Validation, Error Handling, UX of API

- Validation:
  - Data annotations or FluentValidation for create/update DTOs; return 400 with validation problem details.[11] [9]
- Global error handling:
  - Exception-handling middleware mapping exceptions to consistent ProblemDetails (400/401/403/404/500).[1]
- Standardized responses:
  - Uniform envelope structure or at least consistent error format.

Persistence & Performance

- EF Core:
  - Configurations for relationships (OnDelete behavior, indexes on `(TenantId, ProjectId)`, etc.).[11] [1]
  - Use `AsNoTracking` for read-only queries, projection via LINQ to DTOs.[1]
- Caching:
  - Abstraction (interface) for caching “recent projects/boards” and “task counts by status”.
  - In-memory implementation; design so Redis can be plugged in later.[6] [12]

Background Processing & Domain Logic

- Hosted service:
  - Nightly/periodic job to:
    - Mark overdue tasks.
    - Generate per-project task summary (counts per status) and cache/store them.[13] [1]
- Business rules:
  - Limit number of boards/projects per plan (e.g., Free vs Pro tenant).
  - Enforce unique project names per tenant, basic invariants (no task without board, etc.).[4] [3]

Logging, Observability, Security

- Logging:
  - Structured logging with Serilog (or similar).
  - Correlation ID per request in logs.[14] [2]
- Security hardening:
  - Input validation on IDs and search parameters.
  - Review for common issues: SQL injection (EF Core safe usage), not returning sensitive data, secure password hashing.[10] [7]
- Rate limiting:
  - Simple per-IP/per-tenant rate limiting middleware for selected endpoints.[12]

Clean Architecture & Code Quality

- Layering:
  - `Domain`: entities, value objects, domain events.
  - `Application`: interfaces (repositories), DTOs, services/use-cases.
  - `Infrastructure`: EF Core, repositories, cache implementation.
  - `API`: controllers, filters, DI wiring.[2] [15] [1]
- Mapping:
  - Use AutoMapper or manual mappers between domain entities and DTOs.[16] [9]
- Tests:
  - Unit tests for application services (e.g., create task, move task between boards).
  - Integration tests for key endpoints (auth, create project/task, permissions failure).[16] [2]

DevOps & Documentation

- Docker:
  - Dockerfile for API and Docker Compose with API + DB (SQL Server/Postgres).[15] [16]
- Seeding:
  - Seed a demo tenant with sample data for quick testing.[16]
- Documentation (README):
  - Architecture overview diagram (layers + DB).
  - Feature list (multi-tenant, auth, RBAC, caching, background jobs, etc.).[2] [1]
  - Run instructions: `docker compose up`, environment variables, sample requests.

If you want, the next step can be to order these into milestones (MVP, v1, v1.1) so you can track progress across the two weeks.

⁂

1. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures
2. https://ardalis.com/clean-architecture-asp-net-core/
3. https://frontegg.com/guides/multi-tenant-application
4. https://www.commercepundit.com/blog/developing-multi-tenant-applications-in-dotnet-a-guide/
5. https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/service/api-management
6. https://www.apyflux.com/blogs/api-development/multi-tenancy-rest-api
7. https://www.geeksforgeeks.org/c-sharp/asp-net-interview-questions/
8. https://www.iscalepro.com/post/asp-net-core-interview-questions/
9. https://dev.to/thecodewrapper/implementing-a-clean-architecture-in-aspnet-core-6-bdc
10. https://roadmap.sh/questions/backend
11. https://positiwise.com/blog/clean-architecture-net-core
12. https://blog.dreamfactory.com/rate-limiting-vs-throttling-multi-tenant-api-use-cases
13. https://codewithmukesh.com/blog/cqrs-and-mediatr-in-aspnet-core/
14. https://blog.ndepend.com/clean-architecture-for-asp-net-core-solution/
15. https://github.com/ardalis/CleanArchitecture
16. https://github.com/jasontaylordev/CleanArchitecture
17. https://wso2.com/library/articles/2016/08/article-multi-tenant-api-management-with-wso2-api-manager/
18. https://aws.amazon.com/blogs/compute/managing-multi-tenant-apis-using-amazon-api-gateway/
19. https://supertokens.com/blog/multi-tenant-architecture
20. https://thenile.dev/docs/getting-started/examples/task_genius
21. https://www.youtube.com/watch?v=qOwB8PxOqC0
22. https://www.syncloop.com/blogs/enhancing-multitenant-api-workflows-with-syncloop-features.html
23. https://www.linkedin.com/pulse/clean-architecture-asp-net-core-sara-ali-hph5f
24. https://www.youtube.com/watch?v=zw-ZtB1BNI8