گزارش نهایی پروژه مهندسی نرم‌افزار

سامانه مدیریت وظایف **(To-Do) — Full-Stack**

Frontend (React) + Backend (FastAPI) + Docker + Git/PR Workflow

1) مقدمه

این پروژه با هدف پیاده‌سازی چرخه حیات «وظیفه» شامل عملیات CRUD و ارائه یک رابط کاربری زیبا و کاربرپسند اجرا شده است. علاوه بر جنبه فنی، الزامات فرایندی کنترل نسخه با Git و استقرار با Docker نیز مطابق استانداردهای آموزشی رعایت شده‌اند.

2) تعریف مسئله

مسئله اصلی، طراحی و پیاده‌سازی یک Backend استاندارد برای مدیریت وظایف و یک Frontend برای تعامل کاربر با این API است.

2.1) اهداف

- پیاده‌سازی کامل CRUD برای موجودیت Task.
- تعریف قرارداد API دقیق (Status Codeها، Validation، ساختار خطا).
- معماری لایه‌ای و قابل تست برای Backend (Controller/Service/Repository).
- طراحی UI با تم صورتی-پاسی (Pastel) و پشتیبانی از RTL.
- استقرار تکرارپذیر با Docker Compose.
- رعایت فرایند Git با شاخه‌های feature و ادغام از طریق Pull Request.

3) فرضیات و تصمیمات مهندسی

- API: REST + JSON سبک با نسخه‌بندی /api/v1.
- DB: PostgreSQL در محیط Docker Compose.
- حذف: Soft Delete با فیلد deleted_at.
- وضعیت‌های مجاز: TODO، IN_PROGRESS، DONE.
- ذخیره و تبادل زمان‌ها در UTC.

4) نیازمندی‌ها

4.1) نیازمندی‌های عملکردی (Functional)

- FR-01: ایجاد وظیفه (POST /api/v1/tasks).
- FR-02: مشاهده لیست وظایف با صفحه‌بندی و فیلتر (GET /api/v1/tasks).
- FR-03: مشاهده جزئیات یک وظیفه (GET /api/v1/tasks/{id}).
- FR-04: ویرایش وظیفه با (PATCH /api/v1/tasks/{id}) PATCH semantics.
- FR-05: تغییر وضعیت وظیفه (PATCH /api/v1/tasks/{id}/status).
- FR-06: حذف منطقی وظیفه (DELETE /api/v1/tasks/{id}).

4.2 نیازمندی‌های غیرعملکردی (Non-Functional)

- NFR-01: Deployability: اجرای کل سیستم با docker compose up --build.
- NFR-02: Maintainability: جداسازی لایه‌های Backend و رعایت مسئولیت‌ها.
- NFR-03: Testability: وجود تست‌های API و سرویس با pytest.
- NFR-04: Security (حداقلی): Validation و عدم افشای خطاهای داخلی.
- NFR-05: Version Control: توسعه با feature branch و ادغام با PR.

5) Use Case / User Story (خلاصه)

- UC-01: Create Task
- UC-02: List Tasks (pagination + filtering + search)
- UC-03: Update Task
- UC-04: Change Task Status
- UC-05: Delete Task (Soft Delete)

6) UML (متنی)

برای تولید نمودارها می‌توان از PlantUML استفاده کرد. کدهای زیر در حد کافی برای استخراج نمودارهای Use Case و Class و Sequence ارائه می‌شوند.

6.1 Use Case Diagram (PlantUML)

```
@startuml

left to right direction

actor "User/API Client" as U

rectangle "To-Do System" {

  usecase "Create Task" as C

  usecase "List Tasks" as L

  usecase "Get Task" as G

  usecase "Update Task" as U1

  usecase "Change Status" as S

  usecase "Delete Task" as D

}

U --> C

U --> L

U --> G
```

```
U --> U1

U --> S

U --> D

@enduml
```

7) معماری سیستم

معماری Backend به‌صورت لایه‌ای (Clean/Layered) طراحی شده است. Frontend نیز یک SPA است که از طریق قرارداد API با Backend تعامل دارد.

7.1 Backend Layers

- Presentation/API: تعریف endpointها، DTOها و کنترل HTTP.
- Application/Service: پیاده‌سازی use caseها و قوانین دامنه.
- Infrastructure/Repository: دسترسی به DB و عملیات persistence.
- Domain: موجودیت Task و enum وضعیت.

7.2 Frontend

- React + Vite به‌عنوان SPA.
- UI RTL با فونت Vazirmatn.
- تم رنگی صورتی-یاسی (pastel) و طراحی کارت‌ها (glassmorphism).

8) طراحی تفصیلی

8.1 مدل داده (tasks)

- id (UUID/String)
- title (required, max 200)
- description (optional)
- status (enum)
- priority (optional, 1..5)
- due_at (optional datetime)
- created_at / updated_at (UTC)
- deleted_at (for soft delete)

8.2 قرارداد API (خلاصه)

- POST /api/v1/tasks
- GET /api/v1/tasks?limit&offset&status&q
- GET /api/v1/tasks/{id}
- PATCH /api/v1/tasks/{id}
- PATCH /api/v1/tasks/{id}/status
- DELETE /api/v1/tasks/{id}

ساختار پاسخ‌ها به‌صورت Envelope است و خطاهای Validation با 422 و ساختار ثابت شامل trace_id برگردانده می‌شود.

9) پیاده‌سازی

9.1) Backend

Backend با FastAPI و SQLAlchemy پیاده‌سازی شده و از PostgreSQL در Docker Compose استفاده می‌کند. برای سادگی پروژه آموزشی، جدول‌ها در startup ساخته می‌شوند (در پروژه صنعتی، استفاده از Migration توصیه می‌شود).

9.2) Frontend

Frontend با React پیاده‌سازی شده و قابلیت‌های ایجاد/ویرایش/تغییر وضعیت/حذف/فیلتر/جستجو را ارائه می‌کند. آدرس API از طریق متغیر محیطی VITE_API_BASE_URL در زمان build تنظیم می‌شود.

10) تست

تست‌های بک‌اند با pytest پیاده‌سازی شده‌اند و مسیرهای اصلی CRUD و سناریوهای منفی (Validation/NotFound) را پوشش می‌دهند.

11) استقرار و اجرا

کل سیستم با Docker Compose اجرا می‌شود. این روش تکرارپذیر، مستقل از محیط و مناسب تحویل دانشگاهی است.

```
docker compose up --build

# Frontend: http://localhost:3000

# Backend Swagger: http://localhost:8000/docs
```

12) کنترل نسخه با Git (الزامی)

- توسعه روی شاخه‌های feature انجام می‌شود (<feature/<name>).
- ادغام به main فقط از طریق Pull Request انجام می‌شود.
- پیام‌های commit کوتاه، شفاف و تدریجی هستند (Conventional Commits).

13) نتیجه‌گیری

این پروژه یک نمونه کامل و استاندارد از یک سامانه CRUD است که علاوه بر Backend صنعتی، یک Frontend کاربرپسند ارائه می‌دهد و الزامات فرایندی (Git/PR) و استقرار (Docker) را نیز رعایت می‌کند.