

NestJS Task Manager API – Phase 1

A backend Proof of Concept built to understand **NestJS fundamentals**, clean architecture, and **PostgreSQL integration using Docker** (without installing Postgres locally).

This repository intentionally pauses at **Phase 1** to focus on understanding core concepts before enabling advanced NestJS features.



Tech Stack

- Node.js
 - NestJS
 - TypeScript
 - TypeORM
 - PostgreSQL (Docker)
 - Docker & Docker Compose
-



Phase 1 Objectives

- Create a NestJS project using CLI
 - Understand NestJS module structure
 - Implement Controller → Service → Repository flow
 - Integrate PostgreSQL using TypeORM
 - Run PostgreSQL via Docker instead of local installation
 - Build basic CRUD APIs
 - Keep validation concepts understood but **not enabled yet**
-

Project Structure

```
task-manager/
├── src/
│   ├── app.module.ts
│   ├── main.ts
│   └── tasks/
│       ├── dto/
│       │   └── create-task.dto.ts
│       ├── task.entity.ts
│       ├── tasks.controller.ts
│       ├── tasks.module.ts
│       └── tasks.service.ts
└── docker-compose.yml
└── package.json
└── README.md
```

Prerequisites

Make sure the following are installed:

- Node.js (v18+ recommended)
- npm
- Docker
- Docker Compose
- NestJS CLI

Install NestJS CLI:

```
npm install -g @nestjs/cli
```

Create NestJS Project

```
nest new task-manager  
cd task-manager
```

Choose **npm** (or yarn) when prompted.

Install Dependencies

TypeORM + PostgreSQL driver

```
npm install @nestjs/typeorm typeorm pg
```

Validation libraries (used later)

```
npm install class-validator class-transformer
```

Note: Validation is **not enabled in Phase 1**.

DTOs exist for learning and will be activated in Phase 2.

Generate Tasks Module

```
nest generate module tasks  
nest generate controller tasks  
nest generate service tasks
```

This follows the **NestJS modular architecture** pattern.

PostgreSQL Using Docker (No Local Install)

PostgreSQL runs inside Docker.

`docker-compose.yml`

```
version: "3.9"

services:
  postgres:
    image: postgres:16
    container_name: task_postgres
    ports:
      - "5438:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: taskdb
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres -d taskdb"]
      interval: 5s
      timeout: 5s
      retries: 5

volumes:
  postgres_data:
```

Start PostgreSQL

```
docker compose up -d
```

Verify container:

```
docker ps
```

Test database access:

```
docker exec -it task_postgres psql -U postgres -d taskdb
```

TypeORM Configuration

src/app.module.ts

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { TasksModule } from './tasks/tasks.module';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: '127.0.0.1',
      port: 5438,
      username: 'postgres',
      password: 'password',
      database: 'taskdb',
      autoLoadEntities: true,
      synchronize: true, // development only
      retryAttempts: 10,
      retryDelay: 3000,
    }),
    TasksModule,
  ],
})
export class AppModule {}
```

Task Entity

src/tasks/task.entity.ts

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity()
export class Task {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  title: string;

  @Column()
  description: string;

  @Column({ default: false })
```

```
    completed: boolean;  
}
```

Tasks Module

src/tasks/tasks.module.ts

```
import { Module } from '@nestjs/common';  
import { TypeOrmModule } from '@nestjs/typeorm';  
import { TasksController } from './tasks.controller';  
import { TasksService } from './tasks.service';  
import { Task } from './task.entity';  
  
@Module({  
  imports: [TypeOrmModule.forFeature([Task])],  
  controllers: [TasksController],  
  providers: [TasksService],  
})  
export class TasksModule {}
```

Service Layer

src/tasks/tasks.service.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';  
import { InjectRepository } from '@nestjs/typeorm';  
import { Repository } from 'typeorm';  
import { Task } from './task.entity';  
  
@Injectable()  
export class TasksService {  
  constructor(  
    @InjectRepository(Task)  
    private readonly taskRepository: Repository<Task>,  
  ) {}  
  
  findAll(): Promise<Task[]> {  
    return this.taskRepository.find();  
  }  
  
  async findOne(id: number): Promise<Task> {  
    const task = await this.taskRepository.findOne({ where: { id } });  
    if (!task) {  
      throw new NotFoundException(`Task with id ${id} not found`);  
    }  
  }  
}
```

```

        return task;
    }

    create(data: Partial<Task>): Promise<Task> {
        const task = this.taskRepository.create(data);
        return this.taskRepository.save(task);
    }

    async remove(id: number): Promise<void> {
        const result = await this.taskRepository.delete(id);
        if (result.affected === 0) {
            throw new NotFoundException(`Task with id ${id} not found`);
        }
    }
}

```

Controller Layer

src/tasks/tasks.controller.ts

```

import { Controller, Get, Post, Delete, Param, Body } from '@nestjs/common';
import { TasksService } from './tasks.service';
import { CreateTaskDto } from './dto/create-task.dto';
import { Task } from './task.entity';

@Controller('tasks')
export class TasksController {
    constructor(private readonly tasksService: TasksService) {}

    @Get()
    getAll(): Promise<Task[]> {
        return this.tasksService.findAll();
    }

    @Get(':id')
    getOne(@Param('id') id: number): Promise<Task> {
        return this.tasksService.findOne(id);
    }

    @Post()
    create(@Body() dto: CreateTaskDto): Promise<Task> {
        return this.tasksService.create(dto);
    }

    @Delete(':id')
    delete(@Param('id') id: number): Promise<void> {
        return this.tasksService.remove(id);
    }
}

```

```
}
```

Run the Application

```
npm run start:dev
```

Expected output:

```
[Nest] Nest application successfully started
```



Available APIs (Phase 1)

Method	Endpoint	Description
POST	/tasks	Create task
GET	/tasks	List all tasks
GET	/tasks/:id	Get task by ID
DELETE	/tasks/:id	Delete task

Key Learnings from Phase 1

- NestJS modular project structure
- Controller vs Service responsibility separation
- TypeORM Entity & Repository pattern
- Docker-based PostgreSQL setup
- Real-world DB connectivity debugging
- Clean backend foundation before adding validation



Phase 2 Completed

Phase 2 focused on making the application **safe, predictable, and production-ready**.

What was added in Phase 2

- **Global Validation Pipes**
- Enabled **ValidationPipe** to validate all incoming requests
- Invalid requests now fail early with proper **400 Bad Request**

- **DTO-driven Validation**

- `CreateTaskDto` enforces required fields using `@IsDefined` and `@IsNotEmpty`
- `UpdateTaskDto` supports partial updates using `@IsOptional`

- **PATCH Endpoint**

- Added `PATCH /tasks/:id` for partial updates
- Correct REST semantics implemented

- **Environment-based Configuration**

- Integrated `ConfigModule`
- Database credentials moved to `.env`
- No secrets hardcoded in source code

- **Updated Tooling & Documentation**

- Phase 2 Postman collection added
 - README updated to reflect validation, PATCH, and config changes
-

Key Technical Learnings

- Repository pattern via TypeORM
 - Difference between compile-time types and runtime validation
 - Proper use of `async` / `await` (only when needed)
 - DTOs as API contracts
 - Validation as the first line of defense (before DB constraints)
-

Current Status

- Phase 1 completed (core architecture & infrastructure)
- Phase 2 completed (validation, update APIs, config)
- Phase 3 will focus on:
 - Dockerizing the NestJS API
 - CI/CD pipeline using GitHub Actions
 - Production deployment strategy

Author

Santosh Kumar

Backend / Blockchain Engineer

Exploring NestJS with enterprise-grade practices