Task Tracker 開発要求仕様書 (詳細版)

（受験者名またはプロジェクトチーム名）

November 11, 2025

Contents

1 概要 (Introduction)

本ドキュメントは、コーディング試験課題「Task Tracker」の開発に必要な、すべての機能、技術、デプロイ要件を詳細に定義する。

1.1 目的 (Objective)

本課題は、**React (TypeScript)** と **NestJS** を核としたモダンな Web アプリケーションの構築スキル、すなわち、フロントエンド、バックエンド、データベース、およびデプロイの一連のプロセスを理解し、実践する能力を検証することを目的とする。

1.2 アプリケーション概要 (Application Overview)

本アプリケーションは、ログイン認証を備えたシンプルなシングルユーザー向けタスク管理ツールである。ユーザーは、自身のタスクを安全に管理（登録、編集、削除）できる。

1.3 対象読者

- 開発担当者（フロントエンド、バックエンド）
- QA/テスト担当者
- 評価担当者

2 機能要求 (Functional Requirements)

2.1 ユーザー管理機能 (User Management)

1. **サインアップ (Sign Up):**
   - ユーザー名（またはメールアドレス）とパスワードの登録。
   - パスワードは必ずハッシュ化して保存すること。
2. **ログイン (Log In):**
   - ユーザー名（またはメールアドレス）とパスワードによる認証。
   - 認証成功後、クライアント側へ **JWT** を発行する。

2.2 タスクエンティティ (Task Entity)

タスクは以下の必須フィールドを持つ。

フィールド名	データ型	説明
`id`	Integer	主キー（自動採番）
`title`	String (Required)	タスクの概要（例: 企画書作成）
`details`	String (Optional)	タスクの詳細な説明

`status`	Enum (Required)	タスクの現在の状態。後述の 3 種類のみを許容する。
`userId`	Integer	タスクの所有者であるユーザー ID（外部キー）
`createdAt`	DateTime	作成日時（自動設定）
`updatedAt`	DateTime	最終更新日時（自動更新）

2.3 タスク管理機能詳細 (Task Management Details)

(a) **タスク一覧表示:**

- ログイン中のユーザーが所有するすべてのタスクを表示する。
- （推奨）最新の更新日時順などでソートして表示する。

(b) **タスク新規登録:**

- `title` は必須。
- 初期 `status` は ‘TODO‘ とする。
- 登録時、認証情報に基づき `userId` を自動で紐づける。

(c) **タスク編集:**

- `title`, `details`, `status` の全てまたは一部を更新可能。
- `userId` は変更不可。

(d) **タスク削除:**

- 指定されたタスクを完全に削除する。

2.3.1 タスクステータス (Task Status)

ステータスはプルダウンで選択可能とし、以下の 3 種類に限定する。

- **TODO**: 未着手
- **DOING**: 進行中
- **DONE**: 完了

2.4 制約事項 (Constraints)

- **認証必須:** タスク管理に関する全てのエンドポイントは、有効な JWT による認証を必須とする。
- **所有権チェック:** タスクの取得、編集、削除を行う際、リクエストを発行したユーザー ID とタスクの `userId` が一致することをバックエンドで厳密に検証すること。

3 技術仕様と構成 (Technical Specifications & Architecture)

3.1 技術スタック (Technology Stack)

カテゴリ	技術
フロントエンド (Client)	React, TypeScript, Tailwind CSS (推奨), Axios
バックエンド (Server)	NestJS, TypeScript
データベース (DB)	PostgreSQL (Supabase)
ORM	Prisma (推奨) or TypeORM
認証方式	JWT (JSON Web Token)

3.2 ディレクトリ構成 (Directory Structure)

モノレポ構成を採用し、フロントエンドとバックエンドを明確に分離する。

```
root/
├── client/            # React + TypeScript (フロントエンド)
│   ├── src/
│   │   ├── pages/
│   │   ├── components/
│   │   ├── api/           # API クライアント、フックなど
│   │   └── types/         # 共有型定義
│   └── ...
├── server/            # NestJS + TypeScript (バックエンド)
│   ├── src/
│   │   ├── auth/          # 認証モジュール (JWT, Strategy)
│   │   ├── users/         # ユーザーモジュール
│   │   ├── tasks/         # タスクモジュール
│   │   └── main.ts
│   └── prisma/           # Prisma スキーマファイル
├── README.md
└── .env.example
```

3.3 認証フロー (JWT Authentication Flow)

1. **ログイン:**
   - クライアントは ‘/auth/login‘ に認証情報を送信。
   - サーバーは認証後、JWT (Access Token) を生成し、レスポンスボディでクライアントに返す。
2. **タスク操作:**
   - クライアントはタスク関連の API をコールする際、**HTTP** `Authorization` **ヘッダー**に ‘Bearer ¡JWT¿‘ を含める。

- サーバー (NestJS) は **Guard** を利用して JWT を検証し、トークン内のペイロード (`userId` など) をリクエストオブジェクトに挿入する。

3. **認可 (Authorization):**
   - タスク関連のサービス層で、リクエストから取得した `userId` と操作対象のタスクの `userId` を比較し、アクセス権限をチェックする。

4 詳細 API 仕様とデータ定義 (Detailed API Specification & Data Definition)

4.1 データ構造の型定義 (TypeScript Type Definitions)

4.1.1 タスクステータス

```
// client/src/types/task.ts
export type TaskStatus = 'TODO' | 'DOING' | 'DONE';
```

4.1.2 ユーザーとタスクのインターフェース

```
// client/src/types/task.ts
export interface Task {
  id: number;
  title: string;
  details: string | null;
  status: TaskStatus;
  userId: number;
  createdAt: string; // ISO Date String
  updatedAt: string; // ISO Date String
}

// client/src/types/user.ts
export interface User {
  id: number;
  email: string;
}
```

4.2 認証系 API 仕様 (Authentication APIs)

4.2.1 `POST /auth/signup` (サインアップ)

- **リクエストボディ:**

```
{
  "email": "user@example.com",
  "password": "strongpassword123"
}
```

- **レスポンス** (`201 Created`):

```
{
  "message": "User registered successfully"
}
```

4.2.2 POST /auth/login (ログイン)

- **リクエストボディ:** (サインアップと同じ)
- **レスポンス** (200 OK)**:**

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." // JWT
}
```

4.3 タスク管理 API 仕様 (Task Management APIs)

(**注意:** 以下の全ての API は Authorization: Bearer <JWT> ヘッダーを必須とする。)

4.3.1 GET /tasks (タスク一覧取得)

- **リクエストボディ:** なし
- **レスポンス** (200 OK)**:**

```
[
  { "id": 1, "title": "企画書作成", "status": "DOING", ... },
  { "id": 2, "title": "環境構築", "status": "DONE", ... }
]
```

4.3.2 POST /tasks (タスク新規登録)

- **リクエストボディ:**

```
{
  "title": "新しいタスクのアイデア出し",
  "details": "次のプロジェクトの初期アイデアをブレストする。"
}
```

- **レスポンス** (201 Created)**:** (作成されたタスクオブジェクト全体)

```
{
  "id": 3,
  "title": "新しいタスクのアイデア出し",
  "details": "次のプロジェクトの初期アイデアをブレストする。",
  "status": "TODO",
  "userId": 1,
```

```
    "createdAt": "...",
    "updatedAt": "..."
  }
```

4.3.3 PATCH /tasks/:id (タスク編集)

- **リクエストボディ:** (更新したいフィールドのみ)

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

- **レスポンス** (200 OK): (更新後のタスクオブジェクト全体)

4.3.4 DELETE /tasks/:id (タスク削除)

- **リクエストボディ:** なし
- **レスポンス** (200 OK):

```
{
  "message": "Task deleted successfully"
}
```

4.4 DB スキーマ設計 (Prisma Schema Example)

Prisma を使用する場合のスキーマ定義例を示す。

```
// server/prisma/schema.prisma

enum TaskStatus {
  TODO
  DOING
  DONE
}

model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  password  String   // Hashed password
  tasks     Task[]   // Relation to Task model
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Task {
```

```
  id        Int        @id @default(autoincrement())
  title     String
  details   String?
  status    TaskStatus @default(TODO)

  // Relation
  userId    Int
  user      User       @relation(fields: [userId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  @@index([userId])
}
```

5 デプロイ要件 (Deployment Requirements)

5.1 デプロイ構成

対象	推奨サービス	補足
フロントエンド (Client)	Vercel	React アプリケーションのビルドと自動デプロイ。
バックエンド (Server)	Render	NestJS アプリケーションの起動。環境変数による DB 接続。
データベース (DB)	Supabase	PostgreSQL サービスを利用し、接続情報をバックエンドへ提供。

5.2 環境変数 (.env.example)

- **定義の義務:** 必要な環境変数は全て ‘.env.example‘ に明示すること。
- **必須変数:**

```
# --- DB 接続情報 (Supabase/PostgreSQL) ---
DATABASE_URL="postgresql://[USER]:[PASSWORD]@[HOST]:[PORT]/[DATABASE_NAME]?schema

# --- 認証シークレット (NestJS JWT) ---
JWT_SECRET="[YOUR_STRONG_SECRET_KEY]"

# --- CORS 許可オリジン (任意: Render/Vercel URL) ---
CLIENT_URL="https://[VERCEL_DEPLOY_URL]"
```

5.3 提出要件 (Submission Requirements)

- **リポジトリ:** GitHub Public リポジトリ。

- **README.md 記載内容:**
  1. セットアップ手順 (ローカル環境での依存関係インストール、DB 接続、起動方法など)
  2. 使用技術一覧
  3. **デプロイ URL** (フロントエンドと API の両方)
  4. アプリケーションの簡単な説明 (アプリ概要、機能ハイライト)
  5. API 仕様書 (本仕様書の第 4 章をさらに詳細化したもの)

6 評価観点 (Evaluation Criteria)

6.1 評価基準詳細

項目	見たいポイント (詳細)
設計力	• 型定義 (`interface`, `type`) がデータ構造や API の入出力を正確に反映しているか。 • NestJS/React のモジュール分割、コンポーネント化、責務分離が適切に設計されているか。 • フロントエンドとバックエンドで型定義が共有、整合性が保たれているか。
実装力	• 全ての要求機能（CRUD、ステータス更新、認証）が仕様通りに正しく動作するか。 • DB 接続、JWT 認証、認可（所有権チェック）のロジックが確実に機能しているか。
コード品質	• 変数、関数、クラス名の命名規則が一貫し、可読性が高いか。 • 複雑なロジックや非自明な箇所に適切なコメントが付与されているか。 • エラーハンドリング（特に認証失敗、存在しないタスクへのアクセスなど）が適切に行われているか。

UI	• ログイン、タスク一覧、タスク登録・編集の画面が最低限使いやすいデザインか。 • モバイルフレンドリーなレスポンシブ対応がされているか。（推奨） • インタラクション（ボタンのクリック、プルダウン操作など）が直感的か。
デプロイ	• Vercel および Render 上でアプリケーションが安定して動作しているか。 • **README.md に記載された URL** からアクセスし、正常に機能が確認できるか。

7 付録: リポジトリ構成例 (Appendix: Repository Structure)

```
root/
├─── client/            # React（Vercel デプロイ対象）
├─── server/            # NestJS（Render デプロイ対象）
├─── README.md
└─── .env.example
```