NestJSでつくるマルチテナントSaaS

Agenda

- はじめに
- NestJS × マルチテナント × 認証
- NestJS × マルチテナント × MongoDB
- NestJS × マルチテナント × ロギング
- おわりに

はじめに

自己紹介

さわら (@xhiroga) / Twitter

会社紹介

株式会社 justInCaseTechnologies | 保険を変える保険テック会社

プロダクト紹介

保険SaaS基盤: joinsure｜株式会社justInCase：少額短期保険

チーム紹介

技術

チーム紹介

文化

NestJS × マルチテナント × 認証

TL;DR

- 認可トークンを用いたテナントIDの取得を一箇所で行うため、AuthGuardで認証を行う
- テナントIDをLoggerに注入するために、useClass構文を用いる

AuthGuardで認証を行う

- ヘッダーやパス、サブドメインからテナントIDを取得する場合、必ずしもAuthGuardは必要ではない
- OAuthを用いて認可を行い、AccessTokenからテナントIDを取得する場合、AuthGuardは必要
- 今回のデモでは簡略化のためヘッダーからテナントIDを取得する

See AuthGuard

useClass構文を使うことで、LoggerをDIできる

- NestJSでは、GuardのようなMiddleWareもProviderである
- ただし、`app.useGlobalGuards()` で追加した場合、DIのタイミングを逃してしまう
- AppModuleのようなトップレベルのモジュールに対し、特定のInjectionTokenを用いてInjectすることで、DIのタイミングを逃さずにGlobalGuard同様に運用できる

参考

- Custom providers | NestJS - A progressive Node.js framework

See AppModule

まとめ

- AuthGuardで認証を行い、テナントIDをどこからでも取得可能にした
- useClass構文でトップレベルのモジュールにAuthGuardを注入することで、LoggerをDIできた

NestJS × マルチテナント × MongoDB

TL;DR

- MongoDBのDatabaseでテナントを分割した
- ORMにMongooseを選定した
- MongooseのコネクションはDatabaseと1:1
- リクエストスコープでMongooseをInjectするとメモリ不足になる
- Serviceのメソッド実行時、適切なコネクションでModelを生成する

MongoDBのDatabaseでテナントを分割した

- AWS DocumentDBを用いる
  - MongoDB互換のマネージドサービス
- テナントごとにデータを分離する必要がある

MongoDBによるマルチテナント構成

Databaseでテナントを分割した。

単位	Pros	Cons
Cluster	セキュリティが最も高い	インフラ費用、管理コストいずれも高い, テナント数に比例してコストが増加
Database	インフラ費用がテナント数に比例しない, RBACを活用しやすい	DatabaseをまたいだJOINのような処理ができないため、マスターデータとテナント固有データのJOIN処理は工夫が必要
	インフラ費用がテナント数に比	特定のテナントのCollectionのみアク

ORMにMongooseを選定した

MongoDB事例の多さから、手堅くMongooseを選定しました。

	NestJSサポート	Pros	Cons
mongoose	公式Moduleあり	実績が多い, NestJS公式ドキュメントでも取り上げられている, Transactionが使える	公式Moduleはあるが、複数Connectionをサポートしていない
		クラスやデコ	

リクエストスコープでMongooseをInjectするとメモリ不足になる

（意訳）MongoDBとのコネクションをリクエストスコープごとに生成すれば、リクエストごとに適切なテナントに接続できるよ。

- node.js - How to change a Database connection dynamically with Request Scope Providers in Nestjs? - Stack Overflow

リクエストスコープでMongooseをInjectするとメモリ不足になる

デモ:

```
# https://github.com/xhiroga/nestjs-meetup-online1-demo
% yarn dev
% curl localhost:33000/cats
```

Serviceのメソッド実行時、適切なコネクションでModelを生成する

2通りのやり方が存在する。

1. DIを使わない。ConnectionのPoolを自前で持ち、サービスの呼び出し時にModelを生成する。
2. DIを使う。Modelをリクエストスコープで宣言し、ConnectionのPoolをするProviderをInjectする。

る

DIを使わないサンプル

```
@Injectable()
export const CatsService {
    constructor() {
        private readonly connectionProvider: ConnectionProvider;
    }

    async  getCats() {
        const connection = await this.connectionProvider.getConnection();
        const cats = await connection.model('cats').find();
        return cats;
    }
}
```

```
@Injectable()
export class ConnectionProvider{
    // 省略
    getConnection() {
        const tenant = this.request.params.tenantId;
    }
}
```

Serviceのメソッド実行時、適切なコネクションでModelを生成する

DIを使うサンプル

デモ

まとめ

単にMongooseModuleをリクエストスコープで利用するとコネクション数に問題が発生する。
MongoDBのコネクションを自前で管理し、Model生成時に適切に注入することで要件とパフォーマンスを両立できる。

NestJS × マルチテナント × ロギング

TL;DR

- ログにRequestIdとTenantIdを含める
- 全てのErrorをCatchするExceptionsFilterを実装し、エラーを確実にログする

ログにリクエストIDとテナントIDを含める。

- `nestjs-pino` を用いる
- `AuthGuard` でRequestIdを取得する際に、loggerにtenantIdをassignすることで、そのリクエストに対するログにTenantIdを付与できる（厳密なスコープは未検証）

See AuthGuard

全てのErrorをCatchするExceptionsFilterを実装し、エラーを確実にログする

- NestJSは、デフォルトでは全てのエラーをロギングするわけではない
- ドキュメントの通り、全てのエラーをキャッチするExceptionsFilterを実装する
- useClass構文を用いてExceptionsFilterを注入することで、AuthGuardで設定したPinoLoggerを利用できる。

Exception filters | NestJS - A progressive Node.js framework

See AllExceptionsFilter

デモ

まとめ

- `nestjs-pino` でログにRequestIdとTenantIdを含める
- 自前で実装したExceptionsFilterをuseClass構文を用いて注入することで、全てのエラーをtenantId付きでログに出力できる

おわりに

NestJSを用いたマルチテナントSaaSについて

- 今回ご紹介したのは、justInCaseTechnologiesでの取り組みの一部です
- また、私だけでなくチームのメンバーと合わせて取り組んだ成果でもあります
- もっと知りたいという方、ぜひお話したいです！

絶賛採用中！

まずはテックブログをご覧ください💁‍♂️

EOS