RedisService

September 4, 2026

RedisService

Kind: Service

Source: `atloria-monorepo/libs/database/src/lib/redis.service.ts`

`RedisService` is a NestJS service that manages the application's Redis connection lifecycle and provides a small abstraction for common cache operations. It exposes availability checks, raw string operations, and JSON serialization helpers so consumers can use Redis without handling connection details directly.

Methods

Method	Signature	Returns
`onModuleInit`	`onModuleInit()`	`unknown`
`onModuleDestroy`	`onModuleDestroy()`	`unknown`
`getClient`	`getClient()`	`Redis
`isAvailable`	`isAvailable()`	`boolean`
`get`	`get(key: string)`	`Promise<string
`set`	`set(key: string, value: string, ttlSeconds: number)`	`Promise<void>`
`del`	`del(key: string)`	`Promise<void>`
`exists`	`exists(key: string)`	`Promise<boolean>`
`setJson`	`setJson(key: string, value: T, ttlSeconds: number)`	`Promise<void>`
`getJson`	`getJson(key: string)`	`Promise<T
`cached`	`cached(key: string, fetchFn: () => Promise<T>, ttlSeconds: unknown)`	`Promise<T>`
`invalidatePattern`	`invalidatePattern(pattern: string)`	`Promise<void>`

Dependencies

- `ConfigService`

Where it refuses work

- `RedisService` stops the work with an early return when `!this.isConnected || !this.client`, in 5 places.
- `RedisService` stops the work with an early return when `!this.isConnected`.
- `RedisService` stops the work with an early return when `cached`.

When something fails

- `RedisService` handles failure in 8 places: it discards it silently in 4, logs it and continues in 2, and turns it into a return value in 2.

Diagram

```
sequenceDiagram
    participant App as NestJS Application
    participant Service as RedisService
    participant Redis as Redis Server

    App->>Service: onModuleInit()
    Service->>Redis: Connect client
    Redis-->>Service: Connection ready

    App->>Service: setJson(key, value)
    Service->>Redis: SET key JSON.stringify(value)
    Redis-->>Service: OK

    App->>Service: getJson<T>(key)
    Service->>Redis: GET key
    Redis-->>Service: Serialized value
    Service-->>App: Parsed T | null

    App->>Service: onModuleDestroy()
    Service->>Redis: Disconnect client
```

Usage

```
import { Injectable } from '@nestjs/common';
import { RedisService } from '@atloria/database';
```

```ts
interface CachedUser {
  id: string;
  email: string;
  displayName: string;
}

@Injectable()
export class UserCacheService {
  constructor(private readonly redisService: RedisService) {}

  async cacheUser(user: CachedUser): Promise<void> {
    if (!this.redisService.isAvailable()) {
      return;
    }

    await this.redisService.setJson(`user:${user.id}`, user);
  }

  async getCachedUser(userId: string): Promise<CachedUser | null> {
    return this.redisService.getJson<CachedUser>(`user:${userId}`);
  }

  async invalidateUser(userId: string): Promise<void> {
    await this.redisService.del(`user:${userId}`);
  }

  async hasCachedUser(userId: string): Promise<boolean> {
    return this.redisService.exists(`user:${userId}`);
  }
}
```

AI Coding Instructions

- Inject `RedisService` through NestJS dependency injection rather than creating Redis clients directly in feature services.
- Check `isAvailable()` when Redis is optional or when cache failures should not block primary application behavior.
- Use `setJson()` and `getJson<T>()` for structured values; use `set()` and `get()` only for plain string payloads.
- Use consistent, namespaced cache keys such as `user:${id}` or `feature:resource:${id}` to avoid collisions.
- Do not manually connect or disconnect the Redis client; lifecycle handling belongs to `onModuleInit()` and `onModuleDestroy()`.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `AppService` (DEPENDS_ON)
- `AuthService` (DEPENDS_ON)
- `SamlService` (DEPENDS_ON)
- `SsoCodeService` (DEPENDS_ON)
- `CollaborationMetricsService` (DEPENDS_ON)
- `CollaborationService` (DEPENDS_ON)
- `DocsAccessService` (DEPENDS_ON)
- `ReaderMagicLinkService` (DEPENDS_ON)
- `ToursHealingService` (DEPENDS_ON)
- `ToursService` (DEPENDS_ON)
- `BatchScreenshotService` (DEPENDS_ON)
- `FlowReplayService` (DEPENDS_ON)
- `DatabaseModule` (MODULE_PROVIDES)
- `DatabaseModule` (MODULE_EXPORTS)