readme.md for @push.rocks/smartdata

npm version

The ultimate TypeScript-first MongoDB ODM — type-safe decorators, real-time change streams, Lucene-powered search, distributed leader election, and cursor streaming. Built for modern applications that demand performance, correctness, and developer experience.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ Why SmartData?

- □□ **100% Type-Safe** — TC39 Stage 3 decorators, generic filters, and compile-time query validation
- ⚡ **High Performance** — Connection pooling, cursor streaming, and automatic indexing
- □□ **Real-time Ready** — MongoDB Change Streams with RxJS for reactive applications
- □□ **Distributed Systems** — Built-in leader election and task coordination via `@push.rocks/taskbuffer`
- □□ **Security First** — `$where` injection prevention, operator allow-listing, and input sanitization
- □□ **Lucene Search** — Full-text, wildcard, boolean, and range queries out of the box
- □□ **Great DX** — Intuitive API, IntelliSense that just works, and lifecycle hooks

□□ Installation

```
pnpm add @push.rocks/smartdata
```

□□ Requirements

- **Node.js** >= 20.x
- **Deno** >= 2.0 (for Deno projects)
- **MongoDB** >= 5.0
- **TypeScript** >= 5.2 (for TC39 decorator support)

> **Note**: SmartData uses TC39 Stage 3 decorators (the standard). Make sure `experimentalDecorators` is **not** set in your tsconfig.json. Bun is not currently supported as it doesn't implement TC39 decorators yet.

□□ Quick Start

1□ Connect to Your Database

```
import { SmartdataDb } from '@push.rocks/smartdata';

const db = new SmartdataDb({
  mongoDbUrl: 'mongodb://localhost:27017/myapp',
  mongoDbName: 'myapp',
  mongoDbUser: 'username',
  mongoDbPass: 'password',
});

await db.init();
console.log(db.status); // 'connected'
```

2□ Define Your Data Models

```
import {
  SmartDataDbDoc,
```

```
  Collection,
  unI,
  svDb,
  index,
  searchable,
} from '@push.rocks/smartdata';

@Collection(() => db)
class User extends SmartDataDbDoc<User, User> {
  @unI()
  public id!: string;

  @svDb()
  @searchable()
  public username!: string;

  @svDb()
  @searchable()
  @index({ unique: false })
  public email!: string;

  @svDb()
  public status!: 'active' | 'inactive' | 'pending';

  @svDb()
  public tags!: string[];

  @svDb()
  public createdAt: Date = new Date();

  constructor(username: string, email: string) {
    super();
    this.username = username;
    this.email = email;
  }
}
```

3⃣ CRUD Operations

```
// □ Create
const user = new User('johndoe', 'john@example.com');
user.status = 'active';
user.tags = ['developer', 'typescript'];
await user.save();

// □□Read — fully type-safe filters
const foundUser = await User.getInstance({ username: 'johndoe' });
const activeUsers = await User.getInstances({ status: 'active' });

// ✏□ Update
foundUser.email = 'newemail@example.com';
await foundUser.save();

// □□□Delete
await foundUser.delete();
```

□□ Features

□□ Type-Safe Query Filters

SmartData provides a rich, type-safe filtering system supporting all MongoDB operators with full IntelliSense:

```
// Comparison operators
const adults = await User.getInstances({
  age: { $gte: 18, $lt: 65 },
});

// Array operators
const experts = await User.getInstances({
  tags: { $all: ['typescript', 'mongodb'] },
  skills: { $size: 5 },
});

// Logical operators
const complex = await Order.getInstances({
```

```
  $and: [
    { status: 'active' },
    { $or: [{ priority: 'high' }, { value: { $gte: 1000 } }] },
  ],
});

// Deep nested object queries
const users = await User.getInstances({
  profile: {
    settings: {
      notifications: { email: true },
    },
  },
});

// Dot notation
const sameUsers = await User.getInstances({
  'profile.settings.notifications.email': true,
});

// Regex patterns
const gmailUsers = await User.getInstances({
  email: { $regex: '@gmail\\.com$', $options: 'i' },
});
```

Security: The `$where` operator is automatically blocked to prevent NoSQL injection. Unknown operators trigger warnings.

□□ Lucene-Powered Search

Mark fields with `@searchable()` to enable a built-in search engine with automatic compound text indexing:

```
@Collection(() => db)
class Product extends SmartDataDbDoc<Product, Product> {
  @unI() public id!: string;
  @svDb() @searchable() public name!: string;
  @svDb() @searchable() public description!: string;
  @svDb() @searchable() public category!: string;
```

```
  @svDb() public price!: number;
}

// Simple text search across all @searchable fields
const results = await Product.search('laptop');

// Field-specific search
const electronics = await Product.search('category:Electronics');

// Wildcard
const matches = await Product.search('Mac*');

// Boolean operators (AND, OR, NOT)
const query = await Product.search('laptop AND NOT gaming');

// Phrase search
const exact = await Product.search('"MacBook Pro"');

// Range queries
const midRange = await Product.search('price:[100 TO 500]');

// Combined with MongoDB filters and post-fetch validation
const affordable = await Product.search('laptop', {
  filter: { price: { $lte: 1500 } },
  validate: async (p) => p.price > 0,
});
```

□□ Real-Time Change Streams

Watch for database changes with RxJS subjects and EventEmitter support:

```
const watcher = await User.watch(
  { status: 'active' },
  {
    fullDocument: 'updateLookup',
    bufferTimeMs: 100, // optional: buffer changes via RxJS
  },
);
```

```
// RxJS subscription
watcher.changeSubject.subscribe((user) => {
  console.log('User changed:', user);
});

// Or EventEmitter style
watcher.on('change', (user) => {
  console.log('User changed:', user);
});

// Clean up
await watcher.close();
```

□□ Cursor Streaming

Process large datasets without memory pressure:

```
const cursor = await User.getCursor(
  { status: 'active' },
  {
    modifier: (c) => c.sort({ createdAt: -1 }).limit(10000),
  },
);

// Iterate one-by-one
await cursor.forEach(async (user) => {
  await processUser(user);
});

// Or collect into an array
const users = await cursor.toArray();

// Always close when done
await cursor.close();
```

□□ Transactions

Ensure atomic consistency across multiple operations:

```
const session = db.startSession();

try {
  await session.withTransaction(async () => {
    const sender = await User.getInstance({ id: 'user-1' }, { session });
    sender.balance -= 100;
    await sender.save({ session });

    const receiver = await User.getInstance({ id: 'user-2' }, { session });
    receiver.balance += 100;
    await receiver.save({ session });
  });
} finally {
  await session.endSession();
}
```

□□ EasyStore — Type-Safe Key-Value Storage

Built on top of SmartData collections, EasyStore provides simple key-value persistence:

```
interface AppConfig {
  apiKey: string;
  features: { darkMode: boolean; notifications: boolean };
  limits: { maxUsers: number };
}

const config = await db.createEasyStore<AppConfig>('app-config');

// Write
await config.writeKey('features', { darkMode: true, notifications: false });

// Read
const features = await config.readKey('features');
// TypeScript knows: features.darkMode is boolean □

// Read all
```

```
const all = await config.readAll();

// Write multiple keys
await config.writeAll({ apiKey: 'new-key', limits: { maxUsers: 500 } });

// Delete a key
await config.deleteKey('features');

// Wipe the store
await config.wipe();
```

□□ Distributed Coordination

Built-in leader election using MongoDB for coordination, integrating with `@push.rocks/taskbuffer`:

```
import { SmartdataDistributedCoordinator } from '@push.rocks/smartdata';

const coordinator = new SmartdataDistributedCoordinator(db);

// Start coordination — automatic heartbeat and leader election
await coordinator.start();

// Fire distributed task requests
const result = await coordinator.fireDistributedTaskRequest({
  submitterId: 'instance-1',
  requestResponseId: 'unique-id',
  taskName: 'process-payments',
  taskVersion: '1.0.0',
  taskExecutionTime: Date.now(),
  taskExecutionTimeout: 30000,
  taskExecutionParallel: 1,
  status: 'requesting',
});

// Graceful shutdown with leadership handoff
await coordinator.stop();
```

□□ Custom Serialization

Transform data on its way in and out of MongoDB:

```
@Collection(() => db)
class Doc extends SmartDataDbDoc<Doc, Doc> {
  @svDb({
    serialize: (set) => Array.from(set),
    deserialize: (arr) => new Set(arr),
  })
  public tags!: Set<string>;

  @svDb({
    serialize: (date) => date?.toISOString(),
    deserialize: (str) => (str ? new Date(str) : null),
  })
  public scheduledAt!: Date | null;
}
```

□□ Lifecycle Hooks

Add custom logic before and after save/delete:

```
@Collection(() => db)
class Order extends SmartDataDbDoc<Order, Order> {
  @unI() public id!: string;
  @svDb() public items!: Array<{ product: string; quantity: number; price: number }>;
  @svDb() public totalAmount!: number;

  async beforeSave() {
    this.totalAmount = this.items.reduce((s, i) => s + i.price * i.quantity, 0);
  }

  async afterSave() {
    await notificationService.orderUpdated(this.id);
  }

  async beforeDelete() {
    if (this.totalAmount > 0) throw new Error('Cannot delete non-zero orders');
  }
```

```
  async afterDelete() {
    await cache.delete(`order:${this.id}`);
  }
}
```

□□ Indexing

```
@Collection(() => db)
class HighPerformanceDoc extends SmartDataDbDoc<HighPerformanceDoc, HighPerformanceDoc> {
  @unI()
  public id!: string; // Unique index

  @index()
  public userId!: string; // Regular index

  @index({ sparse: true })
  public deletedAt?: Date; // Sparse index — only indexes docs where field exists

  @index({ expireAfterSeconds: 86400 })
  public sessionToken!: string; // TTL index — auto-expires after 24h
}
```

□□ Connection Options

```
const db = new SmartdataDb({
  mongoDbUrl: 'mongodb://localhost:27017',
  mongoDbName: 'myapp',
  mongoDbUser: 'admin',
  mongoDbPass: 's3cret',

  // Connection pool tuning (all optional)
  maxPoolSize: 100,           // Max connections (default: 100)
  maxIdleTimeMS: 300000,      // Close idle connections after 5min (default)
  serverSelectionTimeoutMS: 30000, // Timeout for server selection
  socketTimeoutMS: 30000,     // Socket timeout to prevent hung operations
});
```

□□ Decorators Reference

Decorator	Target	Description
`@Collection(dbGetter)`	Class	Binds a document class to a MongoDB collection
`@managed(managerGetter?)`	Class	Like `@Collection` but controlled by a manager instance
`@unI()`	Field	Marks as unique index + saveable
`@svDb(options?)`	Field	Marks field as saveable, with optional `serialize`/`deserialize`
`@index(options?)`	Field	Creates a regular MongoDB index
`@searchable()`	Field	Enables Lucene-style text search on this field
`@globalSvDb()`	Field	Marks field as globally saveable across all doc types

□□ API Reference

Core Classes

Class	Description
`SmartdataDb`	Database connection, session management, EasyStore factory
`SmartDataDbDoc<T, TImpl>`	Base class for all document models
`SmartdataCollection<T>`	Underlying collection manager (usually accessed indirectly)
`SmartdataDbCursor<T>`	Cursor for streaming large result sets
`SmartdataDbWatcher<T>`	Change stream watcher with RxJS + EventEmitter
`SmartdataDistributedCoordinator`	Leader election and distributed task coordination
`EasyStore<T>`	Type-safe key-value store backed by a collection

Key Static Methods on `SmartDataDbDoc`

Method	Description
`getInstances(filter, opts?)`	Find multiple documents
`getInstance(filter, opts?)`	Find a single document (or `null`)
`getCursor(filter, opts?)`	Get a streaming cursor
`getCount(filter?)`	Count matching documents
`watch(filter, opts?)`	Watch for real-time changes
`search(query, opts?)`	Lucene-style full-text search
`forEach(filter, fn)`	Iterate all matches with a callback
`getNewId(length?)`	Generate a class-prefixed unique ID
`createSearchFilter(luceneQuery)`	Convert Lucene query to MongoDB filter
`getSearchableFields()`	List all `@searchable()` fields

Key Instance Methods on `SmartDataDbDoc`

Method	Description
`save(opts?)`	Insert or update the document
`delete(opts?)`	Delete the document
`updateFromDb()`	Refresh fields from the database
`saveDeep(savedMap?)`	Recursively save referenced documents
`createSavableObject()`	Serialize to a plain object for persistence
`createIdentifiableObject()`	Extract unique index fields for filtering

License and Legal Information

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:10:32 UTC by foss.global Team
Updated 2026-03-28 12:17:18 UTC by foss.global Team