Database Documentation

Technology Stack: MongoDB, NestJS, TypeORM

1. Database Overview

1.1 System Information

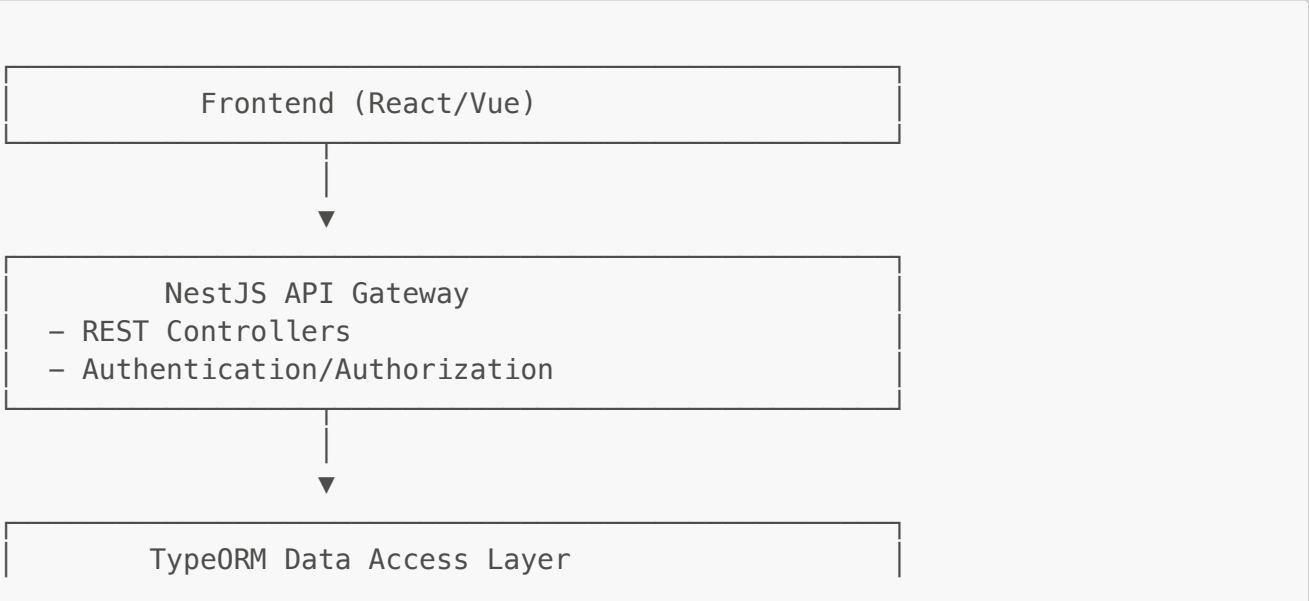
- **Database Type:** MongoDB (NoSQL Document Database)
- **Database Version:** MongoDB 6.0+
- **ORM Framework:** TypeORM (Latest version)
- **Backend Framework:** NestJS
- **Primary Language:** TypeScript/Node.js
- **Architecture Pattern:** Microservices

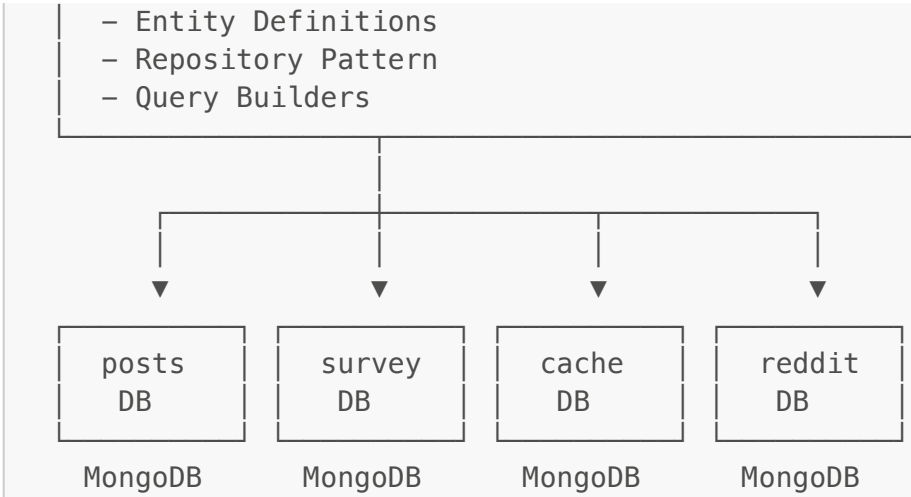
1.2 Database Structure

The Moral Decisions system uses **four separate MongoDB databases**:

Database Name	Purpose	Primary Collections
`posts`	Reddit AITA post data and questions	Question, PostMateData, PostSummary
`survey`	User survey responses and participant data	Answer, Prolific
`cache`	Application-level caching	MoralCache
`reddit`	Reddit API metadata (optional)	N/A

1.3 Architecture Diagram





---

2. Database Connection Information

2.1 Connection Prerequisites

Required Software

```
# Install Node.js (v16+ recommended)
curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -
sudo apt-get install -y nodejs

# Install MongoDB Shell (mongosh)
wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-
key add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
focal/mongodb-org/6.0 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-6.0.list
sudo apt-get update
sudo apt-get install -y mongodb-mongosh

# Install NestJS CLI (for development)
npm install -g @nestjs/cli

# Verify installations
node --version
mongosh --version
nest --version
```

2.2 Connection String Format

Standard MongoDB Connection String

```
mongodb://[username:password@]host[:port]/[database][?options]
```

Production Connection String

```
mongodb://app_user:secure_password@moral-db.example.com:27017/posts?
authSource=admin&retryWrites=true&w=majority
```

Development Connection String

```
mongodb://localhost:27017/posts
```

Connection String with Replica Set

```
mongodb://app_user:password@host1:27017,host2:27017,host3:27017/posts?
replicaSet=rs0&readPreference=primaryPreferred
```

2.3 Environment Configuration

Step 1: Create Environment File

Create a `.env` file in the project root:

```
# Database Connection
MONGODB_URI=mongodb://localhost:27017
DB_POSTS_NAME=posts
DB_SURVEY_NAME=survey
DB_CACHE_NAME=cache
DB_REDDIT_NAME=reddit

# Authentication (Production)
DB_USERNAME=app_user
DB_PASSWORD=your_secure_password
DB_AUTH_SOURCE=admin

# Connection Pool Settings
DB_MAX_POOL_SIZE=50
DB_MIN_POOL_SIZE=10
DB_CONNECTION_TIMEOUT=30000
DB_SOCKET_TIMEOUT=45000

# Logging
DB_LOGGING=true
DB_LOG_LEVEL=info

# Application
```

```
NODE_ENV=development
PORT=3000
```

Step 2: Load Environment Variables

```typescript
// src/config/configuration.ts
import { registerAs } from '@nestjs/config';

export default registerAs('database', () => ({
  uri: process.env.MONGODB_URI,
  posts: process.env.DB_POSTS_NAME,
  survey: process.env.DB_SURVEY_NAME,
  cache: process.env.DB_CACHE_NAME,
  username: process.env.DB_USERNAME,
  password: process.env.DB_PASSWORD,
  authSource: process.env.DB_AUTH_SOURCE,
  maxPoolSize: parseInt(process.env.DB_MAX_POOL_SIZE, 10),
  minPoolSize: parseInt(process.env.DB_MIN_POOL_SIZE, 10),
  connectionTimeout: parseInt(process.env.DB_CONNECTION_TIMEOUT, 10),
  socketTimeout: parseInt(process.env.DB_SOCKET_TIMEOUT, 10),
}));
```

2.4 TypeORM Connection Setup

Configuration for Multiple Databases

```typescript
// src/database/database.module.ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ConfigModule, ConfigService } from '@nestjs/config';

@Module({
  imports: [
    // Posts Database Connection
    TypeOrmModule.forRootAsync({
      name: 'posts',
      imports: [ConfigModule],
      useFactory: (configService: ConfigService) => ({
        type: 'mongodb',
        url: configService.get<string>('database.uri'),
        database: configService.get<string>('database.posts'),
        username: configService.get<string>('database.username'),
        password: configService.get<string>('database.password'),
        authSource: configService.get<string>('database.authSource'),
        entities: [Question, PostMateData, PostSummary],
        synchronize: false, // NEVER true in production
        logging: configService.get<boolean>('database.logging'),
        useUnifiedTopology: true,
```

```
        useNewUrlParser: true,
        maxPoolSize: configService.get<number>('database.maxPoolSize'),
        minPoolSize: configService.get<number>('database.minPoolSize'),
      }),
      inject: [ConfigService],
    }),

    // Survey Database Connection
    TypeOrmModule.forRootAsync({
      name: 'survey',
      imports: [ConfigModule],
      useFactory: (configService: ConfigService) => ({
        type: 'mongodb',
        url: configService.get<string>('database.uri'),
        database: configService.get<string>('database.survey'),
        username: configService.get<string>('database.username'),
        password: configService.get<string>('database.password'),
        authSource: configService.get<string>('database.authSource'),
        entities: [Answer, Prolific],
        synchronize: false,
        logging: configService.get<boolean>('database.logging'),
        useUnifiedTopology: true,
        useNewUrlParser: true,
      }),
      inject: [ConfigService],
    }),

    // Cache Database Connection
    TypeOrmModule.forRootAsync({
      name: 'cache',
      imports: [ConfigModule],
      useFactory: (configService: ConfigService) => ({
        type: 'mongodb',
        url: configService.get<string>('database.uri'),
        database: configService.get<string>('database.cache'),
        entities: [MoralCache],
        synchronize: false,
        logging: false, // Reduce logging for cache operations
        useUnifiedTopology: true,
        useNewUrlParser: true,
      }),
      inject: [ConfigService],
    }),
  ],
})
export class DatabaseModule {}
```

2.5 Direct MongoDB Shell Connection

Connect to Local MongoDB

```
# Connect to default instance
mongosh

# Connect to specific database
mongosh mongodb://localhost:27017/posts

# Connect with authentication
mongosh mongodb://app_user:password@localhost:27017/posts --
authenticationDatabase admin
```

Connect to Remote MongoDB

```
# Standard connection
mongosh "mongodb://username:password@remote-host:27017/posts?
authSource=admin"

# With TLS/SSL
mongosh "mongodb://username:password@remote-host:27017/posts?
authSource=admin&tls=true&tlsCAFile=/path/to/ca.pem"

# Using connection string from environment
mongosh $MONGODB_URI/posts
```

Basic MongoDB Shell Commands

```
// Show all databases
show dbs

// Switch to database
use posts

// Show collections
show collections

// Count documents
db.posts.countDocuments()

// Find sample document
db.posts.findOne()

// Show database statistics
db.stats()

// Check connection status
db.adminCommand({ connectionStatus: 1 })
```

2.6 Application Code Connection Examples

Using Repository Pattern (Recommended)

```
// src/posts/posts.service.ts
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { MongoRepository } from 'typeorm';
import { Question } from './entities/question.entity';

@Injectable()
export class PostsService {
  constructor(
    @InjectRepository(Question, 'posts')
    private readonly questionRepository: MongoRepository<Question>,
  ) {}

  async findAll(): Promise<Question[]> {
    return this.questionRepository.find();
  }

  async findById(id: string): Promise<Question> {
    return this.questionRepository.findOne({ where: { _id: id } });
  }
}
```

Direct MongoDB Native Driver Connection

```
// src/database/mongo.service.ts
import { Injectable, OnModuleInit } from '@nestjs/common';
import { MongoClient, Db } from 'mongodb';
import { ConfigService } from '@nestjs/config';

@Injectable()
export class MongoService implements OnModuleInit {
  private client: MongoClient;
  private postsDb: Db;
  private surveyDb: Db;
  private cacheDb: Db;

  constructor(private configService: ConfigService) {}

  async onModuleInit() {
    const uri = this.configService.get<string>('database.uri');

    this.client = new MongoClient(uri, {
      maxPoolSize: 50,
      minPoolSize: 10,
      serverSelectionTimeoutMS: 5000,
```

```
        socketTimeoutMS: 45000,
      });

      await this.client.connect();

      this.postsDb = this.client.db('posts');
      this.surveyDb = this.client.db('survey');
      this.cacheDb = this.client.db('cache');

      console.log('✅ MongoDB connected successfully');
    }

    getPostsDb(): Db {
      return this.postsDb;
    }

    getSurveyDb(): Db {
      return this.surveyDb;
    }

    getCacheDb(): Db {
      return this.cacheDb;
    }

    async disconnect() {
      await this.client.close();
    }
  }
```

2.7 Connection Troubleshooting

Common Connection Issues

Error	Cause	Solution
`MongoServerError: bad auth`	Invalid credentials	Verify username/password in `.env`, check `authSource`
`MongoNetworkError: connection timeout`	Network/firewall issue	Check MongoDB is running, verify host/port, check firewall rules
`MongooseError: Can't call \openUri() on an active connection`	Multiple connection attempts	Ensure only one connection is initialized
`TypeError: Cannot read property 'db' of undefined`	Connection not established	Wait for connection before querying, use `onModuleInit`
`MongoServerError: not authorized`	Insufficient permissions	Grant proper roles to database user

Connection Diagnostics

```
// src/health/database.health.ts
import { Injectable } from '@nestjs/common';
import { HealthIndicator, HealthIndicatorResult, HealthCheckError } from
'@nestjs/terminus';
import { InjectConnection } from '@nestjs/typeorm';
import { Connection } from 'typeorm';

@Injectable()
export class DatabaseHealthIndicator extends HealthIndicator {
  constructor(
    @InjectConnection('posts')
    private postsConnection: Connection,
    @InjectConnection('survey')
    private surveyConnection: Connection,
  ) {
    super();
  }

  async isHealthy(key: string): Promise<HealthIndicatorResult> {
    try {
      const postsConnected = this.postsConnection.isConnected;
      const surveyConnected = this.surveyConnection.isConnected;

      if (postsConnected && surveyConnected) {
        return this.getStatus(key, true, {
          posts: 'up',
          survey: 'up',
        });
      }

      throw new HealthCheckError('Database check failed', {
        posts: postsConnected ? 'up' : 'down',
        survey: surveyConnected ? 'up' : 'down',
      });
    } catch (error) {
      throw new HealthCheckError('Database check failed', error);
    }
  }
}
```

Testing Connection

```
# Test database connection
npm run test:db:connection

# Health check endpoint
curl http://localhost:3000/health

# Check specific database
mongosh mongodb://localhost:27017/posts --eval "db.adminCommand('ping')"
```

2.8 Connection Security Best Practices

1. **Never hardcode credentials** - Always use environment variables
2. **Use connection pooling** - Reuse connections, don't create new ones per request
3. **Enable TLS/SSL in production** - Encrypt data in transit
4. **Implement retry logic** - Handle transient connection failures
5. **Monitor connection pool** - Track active/idle connections
6. **Use read replicas** - Distribute read load across replica set
7. **Set appropriate timeouts** - Prevent hanging connections
8. **Validate environment config** - Fail fast on missing/invalid configuration

---

3. Database Modification Procedures - 如何修改

3.1 Schema Modification Overview

MongoDB is schema-less, but TypeORM entities define the application-level schema. Modifications involve:

1. Updating TypeORM entity definitions
2. Creating migration scripts
3. Testing changes in development
4. Applying to production

3.2 Modifying Entity Definitions

Adding a New Field to Existing Collection

Example: Adding `createdBy` field to Question entity

```
// src/posts/entities/question.entity.ts (BEFORE)
import { Entity, ObjectIdColumn, ObjectId, Column } from 'typeorm';

@Entity('posts')
export class Question {
  @ObjectIdColumn()
  _id: string;

  @Column()
  title: string;

  @Column()
  selftext: string;

  @Column()
  very_certain_YA: number;

  @Column()
  very_certain_NA: number;
}
```

```
// src/posts/entities/question.entity.ts (AFTER)
import { Entity, ObjectIdColumn, ObjectId, Column } from 'typeorm';

@Entity('posts')
export class Question {
  @ObjectIdColumn()
  _id: string;

  @Column()
  title: string;

  @Column()
  selftext: string;

  @Column()
  very_certain_YA: number;

  @Column()
  very_certain_NA: number;

  // NEW FIELD
  @Column({ nullable: true })
  createdBy?: string;

  @Column({ type: 'date', nullable: true })
  createdAt?: Date;
}
```

Creating a New Entity/Collection

Example: Adding UserSession collection

```
// src/survey/entities/user-session.entity.ts
import { Entity, ObjectIdColumn, ObjectId, Column, Index } from 'typeorm';

@Entity('user_sessions')
@Index(['prolificId', 'studyId'])
export class UserSession {
  @ObjectIdColumn()
  _id: ObjectId;

  @Column()
  @Index()
  prolificId: string;

  @Column()
  studyId: number;
```

```
  @Column()
  sessionToken: string;

  @Column({ type: 'date' })
  startedAt: Date;

  @Column({ type: 'date', nullable: true })
  expiresAt?: Date;

  @Column({ default: true })
  isActive: boolean;

  @Column({ type: 'object', nullable: true })
  metadata?: Record<string, any>;
}
```

Register entity in module:

```
// src/survey/survey.module.ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UserSession } from './entities/user-session.entity';
import { Answer } from './entities/answer.entity';
import { Prolific } from './entities/prolific.entity';

@Module({
  imports: [
    TypeOrmModule.forFeature([Answer, Prolific, UserSession], 'survey'),
  ],
  // ...
})
export class SurveyModule {}
```

3.3 Data Migration Scripts

Creating a Migration

```
# Generate migration file
npm run migration:generate -- -n AddCreatedByToQuestion

# Create empty migration
npm run migration:create -- -n AddUserSessionCollection
```

Migration File Structure

```
// src/migration/1696723456789-AddCreatedByToQuestion.ts
import { MigrationInterface } from 'typeorm';
import { MongoClient } from 'mongodb';

export class AddCreatedByToQuestion1696723456789 implements
MigrationInterface {
  public async up(): Promise<void> {
    const client = new MongoClient(process.env.MONGODB_URI);
    await client.connect();

    try {
      const db = client.db('posts');
      const collection = db.collection('posts');

      // Add createdBy field to all existing documents
      await collection.updateMany(
        { createdBy: { $exists: false } },
        {
          $set: {
            createdBy: 'system',
            createdAt: new Date(),
          },
        }
      );

      console.log('✅ Migration completed: Added createdBy field');
    } finally {
      await client.close();
    }
  }

  public async down(): Promise<void> {
    const client = new MongoClient(process.env.MONGODB_URI);
    await client.connect();

    try {
      const db = client.db('posts');
      const collection = db.collection('posts');

      // Remove createdBy field
      await collection.updateMany(
        {},
        {
          $unset: {
            createdBy: '',
            createdAt: '',
          },
        }
      );

      console.log('✅ Rollback completed: Removed createdBy field');
    } finally {
      await client.close();
```

```
    }
  }
}
```

Running Migrations

```
# Run all pending migrations
npm run migration:run

# Revert last migration
npm run migration:revert

# Show migration status
npm run migration:show
```

3.4 Modifying Data

Bulk Update Operations

```
// src/posts/posts.service.ts
async updateVerdictPercentages(): Promise<void> {
  const questions = await this.questionRepository.find();

  for (const question of questions) {
    const total = question.very_certain_YA + question.very_certain_NA;

    if (total > 0) {
      question.YA_percentage = (question.very_certain_YA / total) * 100;
      question.NA_percentage = (question.very_certain_NA / total) * 100;

      await this.questionRepository.save(question);
    }
  }

  console.log(`Updated ${questions.length} questions`);
}
```

Using MongoDB Aggregation for Updates

```
async recalculateStatistics(): Promise<void> {
  const db = this.connection.db;

  const result = await db.collection('posts').aggregate([
    {
      $set: {
        YA_percentage: {
```

```
          $multiply: [
            { $divide: ['$very_certain_YA', { $add: ['$very_certain_YA',
'$very_certain_NA'] }] },
            100
          ]
        },
        NA_percentage: {
          $multiply: [
            { $divide: ['$very_certain_NA', { $add: ['$very_certain_YA',
'$very_certain_NA'] }] },
            100
          ]
        }
      }
    },
    {
      $out: 'posts'  // Replace collection with updated documents
    }
  ]).toArray();

  console.log('Statistics recalculated');
}
```

3.5 Index Management

Creating Indexes

```
// Create index programmatically
async createIndexes(): Promise<void> {
  const db = this.connection.db;

  // Single field index
  await db.collection('posts').createIndex({ title: 1 });

  // Compound index
  await db.collection('answer').createIndex(
    { prolificId: 1, studyId: 1 },
    { unique: true }
  );

  // Text index for full-text search
  await db.collection('posts').createIndex(
    { title: 'text', selftext: 'text' },
    { weights: { title: 10, selftext: 1 } }
  );

  // TTL index (auto-delete expired documents)
  await db.collection('MoralCache').createIndex(
    { expiresAt: 1 },
    { expireAfterSeconds: 0 }
  );
```

```
  console.log('✅ Indexes created successfully');
}
```

Viewing and Managing Indexes

```
// MongoDB shell commands

// List all indexes
db.posts.getIndexes()

// Create index
db.posts.createIndex({ "verdict": 1 })

// Drop index
db.posts.dropIndex("verdict_1")

// Rebuild all indexes
db.posts.reIndex()

// Check index usage statistics
db.posts.aggregate([{ $indexStats: {} }])
```

3.6 Collection Operations

Creating a New Collection

```
// MongoDB shell
db.createCollection('user_sessions', {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: ['prolificId', 'studyId', 'sessionToken'],
      properties: {
        prolificId: { bsonType: 'string' },
        studyId: { bsonType: 'int' },
        sessionToken: { bsonType: 'string' },
        startedAt: { bsonType: 'date' },
        expiresAt: { bsonType: 'date' },
        isActive: { bsonType: 'bool' }
      }
    }
  }
})
```

Renaming a Collection

```
// MongoDB shell
db.old_collection_name.renameCollection('new_collection_name')
```

Dropping a Collection

```
// MongoDB shell - CAUTION: This deletes all data!
db.collection_name.drop()
```

3.7 Backup Before Modifications

Always backup before making schema or data changes:

```
# Backup specific database
mongodump --uri="mongodb://localhost:27017" --db=posts --
out=/backup/$(date +%Y%m%d)

# Backup specific collection
mongodump --uri="mongodb://localhost:27017" --db=posts --collection=posts
--out=/backup/posts_backup

# Backup with compression
mongodump --uri="mongodb://localhost:27017" --db=posts --gzip --
out=/backup/posts_compressed
```

3.8 Testing Modifications

Development Testing Workflow

1. **Local Testing**

```
# Use separate test database
MONGODB_URI=mongodb://localhost:27017 DB_POSTS_NAME=posts_test npm run
test
```

2. **Migration Testing**

```
# Apply migration
npm run migration:run

# Test application
npm run test:e2e
```

```
# Rollback if issues found
npm run migration:revert
```

3. **Data Validation**

```
// Validate data after modification
async validateQuestions(): Promise<void> {
  const invalidQuestions = await this.questionRepository.find({
    where: {
      $or: [
        { title: { $exists: false } },
        { selftext: { $exists: false } },
        { YA_percentage: { $lt: 0 } },
        { NA_percentage: { $gt: 100 } },
      ],
    },
  });

  if (invalidQuestions.length > 0) {
    console.error(`Found ${invalidQuestions.length} invalid questions`);
    throw new Error('Data validation failed');
  }
}
```

3.9 Production Deployment Checklist

- [ ] Backup current database
- [ ] Test migration in staging environment
- [ ] Review migration script for performance impact
- [ ] Schedule maintenance window if needed
- [ ] Prepare rollback procedure
- [ ] Apply migration during low-traffic period
- [ ] Monitor application logs for errors
- [ ] Validate data integrity post-migration
- [ ] Update documentation
- [ ] Notify team of completion

---

4. Database Content Documentation

4.1 Database: `posts` - Reddit AITA Post Data

Collection 4.1.1: `posts` (Question Entity)

Purpose: Stores Reddit AITA (Am I The Asshole) posts used as survey questions

Document Count: ~2,500 posts **Average Size:** 2.5 KB per document **Total Size:** ~6.2 MB

Complete Schema:

Field Name	Data Type	Nullable	Default	Description
`_id`	String	No	-	Reddit post ID (e.g., "abc123xyz")
`title`	String	No	-	Reddit post title (max 300 characters)
`selftext`	String	No	-	Full post content/body text
`very_certain_YA`	Number	No	0	Count of "very certain You're Asshole" votes
`very_certain_NA`	Number	No	0	Count of "very certain Not Asshole" votes
`YA_percentage`	Number	No	0	Percentage of YA votes (0-100)
`NA_percentage`	Number	No	0	Percentage of NA votes (0-100)
`original_post_YA_top_reasonings`	String[]	No	[]	Top 3-5 YA reasoning comments
`original_post_NA_top_reasonings`	String[]	No	[]	Top 3-5 NA reasoning comments
`count`	Number[]	No	[]	Vote distribution: [YTA, NTA, ESH, NAH, INFO] counts

Indexes:

- `_id` (Primary Key, unique)
- `{ title: "text", selftext: "text" }` (Full-text search)

Sample Document:

```
{
  "_id": "abc123xyz",
  "title": "AITA for refusing to babysit my sister's kids?",
  "selftext": "My sister (32F) asked me (28F) to babysit her three kids
(ages 3, 5, 7) every weekend...",
  "very_certain_YA": 45,
  "very_certain_NA": 389,
  "YA_percentage": 10.4,
  "NA_percentage": 89.6,
  "original_post_YA_top_reasonings": [
    "You made a commitment to your sister",
    "Family should help each other",
```

```
      "It's only one weekend"
    ],
    "original_post_NA_top_reasonings": [
      "You have no obligation to provide free childcare",
      "Your time is valuable",
      "She's taking advantage of you",
      "Setting boundaries is healthy"
    ],
    "count": [45, 389, 12, 8, 5]
  }
```

Query Examples:

```
// Find posts with high NA consensus
const consensusPosts = await questionRepository.find({
  where: { NA_percentage: { $gte: 80 } },
  take: 20
});

// Search posts by keyword
const relationshipPosts = await questionRepository.find({
  where: { $text: { $search: "relationship boyfriend girlfriend" } }
});
```

Content Maintenance Workflow (Posts Collection):

1. **Confirm Target Study** – Open the survey page and capture the `studyId` query parameter (e.g., `...?studyId=2`). Each `studyId` maps to a single question document and is used by the backend when selecting unshared questions (`src/service/survey.service.ts`).
2. **Review Existing Data** – Connect via `mongosh "<DATABASE_URL>"`, switch to the survey database (`use survey`), and inspect the current document with `db.posts.find({ _id: "<id>" }).pretty()`. Export a backup before editing to preserve a rollback point.
3. **Update / Insert Document** – Required fields include `_id`, `title`, `selftext`, certainty metrics (`very_certain_YA`, `very_certain_NA`, `YA_percentage`, `NA_percentage`), reasoning arrays (`original_post_YA_top_reasonings`, `original_post_NA_top_reasonings`), and `count` (object keyed by study ID or array aligned to the configured studies). Apply changes with an `updateOne`/`insertOne`, for example:

   ```
   db.posts.updateOne(
     { _id: "aita_001" },
     {
       $set: {
         title: "New title",
         selftext: "Full scenario text...",
         very_certain_YA: 0.42,
         very_certain_NA: 0.31,
         YA_percentage: 0.58,
         NA_percentage: 0.42,
   ```

```
        original_post_YA_top_reasonings: ["Reason A", "Reason B"],
        original_post_NA_top_reasonings: ["Reason C", "Reason D"],
        count: { "1": 0, "2": 0, "3": 0, "4": 0, "5": 0 }
      }
    },
    { upsert: true }
  );
```

Keep `count[studyId]` synchronized with valid study IDs to avoid “studyId out of range” errors.

4. **Refresh Static Frontend Assets** – Update training examples, attention checks, and Likert items in `moral-survey/src/js/litw/litw.data.2.0.0.js`, adjust templates under `moral-survey/src/templates/`, and revise translations (`moral-survey/src/moral-survey-1/i18n/*.json`) when copy changes. Rebuild the relevant bundle from the survey subdirectory (e.g., `cd moral-survey/src/moral-survey-1 && npx webpack --config webpack.config.js --mode production`).
5. **Restart Backend Services** – Bounce the NestJS server (or reload the process in PM2/systemd) so configuration and caches refresh, especially after environment variable changes.
6. **Validate End-to-End** – Hit `GET /survey/question?studyId=<id>` to confirm the updated document, load the survey page (`http://localhost:8080/moral-survey-1/index.html?studyId=<id>` or production equivalent), and submit a test response. Verify persistence via `db.answer.find().sort({ _id: -1 }).limit(1)`.
7. **Document & Deploy** – Commit backend/frontend changes with descriptive messages, update shared scripts or runbooks, and deploy through the standard release pipeline before repeating validation in staging/production.

Collection 4.1.2: `all` (PostMateData Entity)

Purpose: Comprehensive Reddit post metadata and analytics

Document Count: ~15,000 posts **Average Size:** 1.8 KB per document **Total Size:** ~27 MB

Complete Schema:

Field Name	Data Type	Description
`_id`	String	Reddit post unique identifier
`year`	Integer	Post creation year (e.g., 2023)
`month`	Integer	Post creation month (1-12)
`day`	Integer	Post creation day (1-31)
`verdict`	String	Final verdict: YTA/NTA/ESH/NAH
`num_words`	Integer	Word count in post body
`label_entropy`	Double	Verdict certainty (0=certain, higher=divided)
`score`	Integer	Reddit post score (upvotes - downvotes)

Field Name	Data Type	Description
YTA	Integer	"You're the Asshole" vote count
NTA	Integer	"Not the Asshole" vote count
ESH	Integer	"Everyone Sucks Here" vote count
NAH	Integer	"No Assholes Here" vote count
INFO	Integer	"Need More Info" vote count
title	String	Post title text
topic_1	String	Primary topic (e.g., "family", "relationship")
topic_1_p	Double	Primary topic probability (0.0-1.0)
topic_2	String	Secondary topic
topic_2_p	Double	Secondary topic probability
topic_3	String	Tertiary topic
topic_3_p	Double	Tertiary topic probability
topic_4	String	Quaternary topic
year_r	Integer	Response/verdict year
month_r	Integer	Response/verdict month
day_r	Integer	Response/verdict day
hour	Integer	Response hour (0-23)
minute	Integer	Response minute (0-59)
second	Integer	Response second (0-59)
num_words_r	Integer	Response comment word count
num_comments	Integer	Total comment count on post
score_r	Integer	Response comment score
flair	String	Post flair category
verdict_r	String	Response verdict text
resolved_verdict	String	Final resolved verdict
v_id	String	Verdict comment ID
v_year	Integer	Verdict assigned year
v_month	Integer	Verdict assigned month
v_day	Integer	Verdict assigned day
v_hour	Integer	Verdict assigned hour

Field Name	Data Type	Description
`v_minute`	Integer	Verdict assigned minute
`v_second`	Integer	Verdict assigned second
`v_words`	Integer	Verdict comment word count
`v_score`	Integer	Verdict comment score

Indexes:

- `_id` (Primary Key)
- `{ verdict: 1 }`
- `{ year: 1, month: 1, day: 1 }` (Compound date index)
- `{ topic_1: 1 }`

Typical Topics:

- Family conflicts
- Relationship issues
- Workplace disputes
- Parenting dilemmas
- Financial disagreements
- Wedding drama
- Property/housing disputes

Collection 4.1.3: `post_summary` (PostSummary Entity)

Purpose: Lightweight post summaries for quick listing and search

Document Count: ~2,500 **Average Size:** 800 bytes **Total Size:** ~2 MB

Complete Schema:

Field Name	Data Type	Description
`id`	String	Post identifier (matches `posts._id`)
`title`	String	Post title
`verdict`	String	Final verdict (YTA/NTA/ESH/NAH)
`YTA`	Number	YTA vote count
`NTA`	Number	NTA vote count
`selftext`	String	Truncated post content (first 500 chars)
`commentCount`	Number	Total number of comments
`topics`	String[]	Array of associated topics (e.g., ["family", "money"])

Sample Document:

```
{
  "id": "xyz789",
  "title": "AITA for not attending my brother's wedding?",
  "verdict": "NTA",
  "YTA": 23,
  "NTA": 412,
  "selftext": "My brother is getting married next month, but he
specifically told me my partner isn't welcome...",
  "commentCount": 156,
  "topics": ["family", "wedding", "relationship"]
}
```

4.2 Database: `survey` - User Responses and Demographics

Collection 4.2.1: `answer` (Answer Entity)

Purpose: Complete survey responses from participants

Document Count: ~1,850 responses **Average Size:** 8 KB per document **Total Size:** ~14.8 MB

Complete Schema:

Field Name	Data Type	Nullable	Description
`_id`	ObjectId	No	MongoDB unique identifier
`prolificId`	String	No	Prolific platform participant ID (e.g., "5f9a1b2c3d4e5f6a7b8c9d0e")
`studyId`	Number	No	Study identifier (1-5 for different study waves)
`answerDetail`	AnswerItem[]	No	Array of individual question responses
`comment`	String	Yes	General participant feedback/comment
`decisionMaking`	Number[]	No	25-item decision-making style questionnaire (values 1-5)
`personalityChoice`	Number[]	No	15-item personality questionnaire (values 1-7)
`time`	BigInt	No	Survey completion timestamp (Unix milliseconds)

Nested Type: AnswerItem Structure

Field Name	Data Type	Description
`_id`	ObjectId	Item unique identifier
`questionId`	String	Reference to Question._id (Reddit post ID)
`individualAnswer`	AnswerObject	Participant's personal judgment

Field Name	Data Type	Description
groupAnswer	AnswerObject	Participant's prediction of group consensus
comment	String	Question-specific comment (optional)

AnswerObject Structure:

Field Name	Data Type	Range	Description
isAsshole	Boolean	true/false	Asshole determination (true = YTA, false = NTA)
rating	Number	1-5	Severity rating (1=minor, 5=severe)

Sample Document:


```
{
  "_id": "507f1f77bcf86cd799439011",
  "prolificId": "5f9a1b2c3d4e5f6a7b8c9d0e",
  "studyId": 3,
  "answerDetail": [
    {
      "_id": "507f191e810c19729de860ea",
      "questionId": "abc123xyz",
      "individualAnswer": {
        "isAsshole": false,
        "rating": 2
      },
      "groupAnswer": {
        "isAsshole": false,
        "rating": 3
      },
      "comment": "I think most people would agree this is justified"
    },
    {
      "_id": "507f191e810c19729de860eb",
      "questionId": "def456uvw",
      "individualAnswer": {
        "isAsshole": true,
        "rating": 4
      },
      "groupAnswer": {
        "isAsshole": true,
        "rating": 5
      },
      "comment": ""
    }
  ],
  "comment": "Very interesting survey, made me think about my own moral
judgments",
  "decisionMaking": [3, 4, 2, 5, 3, 3, 4, 2, 3, 4, 3, 2, 4, 3, 3, 4, 2, 3,
5, 4, 3, 2, 3, 4, 3],
  "personalityChoice": [4, 5, 3, 6, 4, 3, 5, 4, 3, 5, 4, 6, 3, 4, 5],
```

```
  "time": 1696723456789
}
```

Indexes:

- `_id` (Primary Key)
- `{ prolificId: 1, studyId: 1 }` (Compound, unique)
- `{ prolificId: 1 }`
- `{ time: 1 }`

Decision-Making Scale (25 items):

- Questions about: intuition vs. analysis, speed vs. deliberation, emotional vs. rational thinking
- Scale: 1=Strongly Disagree to 5=Strongly Agree

Personality Scale (15 items):

- Big Five personality traits adaptation
- Scale: 1=Not at all like me to 7=Exactly like me

Collection 4.2.2: `prolifics` (Prolific Entity)

Purpose: Participant demographic and platform data

Document Count: ~1,950 participants **Average Size:** 400 bytes **Total Size:** ~780 KB

Complete Schema:

Field Name	Data Type	Nullable	Description
`_id`	ObjectId	No	MongoDB unique identifier
`prolificId`	String	No	Unique Prolific participant ID (PRIMARY KEY)
`age`	Number	No	Participant age (18-99)
`country`	String	No	Country code (e.g., "US", "GB", "AU")
`frequentUser`	Boolean	No	Whether participant is frequent Reddit user
`language`	String	No	Primary language (e.g., "English", "Spanish")
`takenBefore`	Boolean[]	No	Array of 5 booleans indicating previous study participation
`visitSubreddit`	Boolean	No	Whether participant has visited r/AmItheAsshole

Sample Document:

```
{
  "_id": "507f1f77bcf86cd799439012",
```

```
  "prolificId": "5f9a1b2c3d4e5f6a7b8c9d0e",
  "age": 28,
  "country": "US",
  "frequentUser": true,
  "language": "English",
  "takenBefore": [false, false, true, false, false],
  "visitSubreddit": true
}
```

Indexes:

- `_id` (Primary Key)
- `{ prolificId: 1 }` (Unique)
- `{ country: 1 }`

Demographics Summary (Current Data):

- **Age Range:** 18-75 (mean: 32.4, median: 29)
- **Countries:** 15 countries (US: 68%, UK: 18%, AU: 7%, CA: 4%, Other: 3%)
- **Languages:** Primarily English (94%), Spanish (3%), Other (3%)
- **Reddit Users:** 73% frequent users, 89% visited r/AITA
- **Returning Participants:** 24% participated in previous studies

---

4.3 Database: `cache` - Application Cache

Collection 4.3.1: `MoralCache` (Cache Entity)

Purpose: Application-level caching for API responses and computed results

Document Count: Variable (~500-2,000) **Average Size:** 2 KB **Total Size:** ~1-4 MB (fluctuates)

Complete Schema:

Field Name	Data Type	Nullable	Description
`_id`	ObjectId	No	MongoDB unique identifier
`key`	String	No	Cache key (unique, e.g., "posts_list_page_1")
`value`	String	No	Cached value (JSON stringified)
`expiresAt`	Date	No	Expiration timestamp (auto-delete via TTL index)
`hit`	Number	Yes	Cache hit counter for analytics

Indexes:

- `_id` (Primary Key)
- `{ key: 1 }` (Unique)
- `{ expiresAt: 1 }` (TTL index with expireAfterSeconds: 0)

Sample Document:

```
{
  "_id": "507f1f77bcf86cd799439013",
  "key": "posts_list_page_1_limit_20",
  "value": "[{\"_id\":\"abc123\",\"title\":\"AITA
for...\",\"verdict\":\"NTA\"}...]",
  "expiresAt": "2025-10-07T18:30:00.000Z",
  "hit": 47
}
```


Common Cache Keys:

- `posts_list_page_{page}_limit_{limit}` - Paginated post listings
- `post_detail_{postId}` - Individual post details
- `user_answers_{prolificId}` - User's survey responses
- `statistics_summary` - Global statistics
- `topic_analysis_{topic}` - Topic-specific analytics

Cache Strategy:

- **TTL:** Most cache entries expire after 1 hour
- **Invalidation:** Manually cleared when underlying data changes
- **Hit Tracking:** `hit` counter helps identify hot cache entries

---

4.4 Database: `reddit` - Reddit API Metadata (Optional)

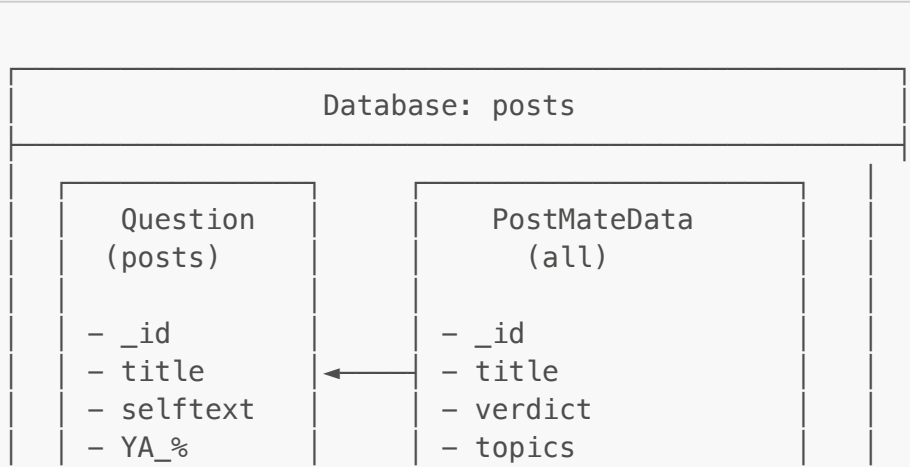
Purpose: Stores raw Reddit API responses and rate limit information

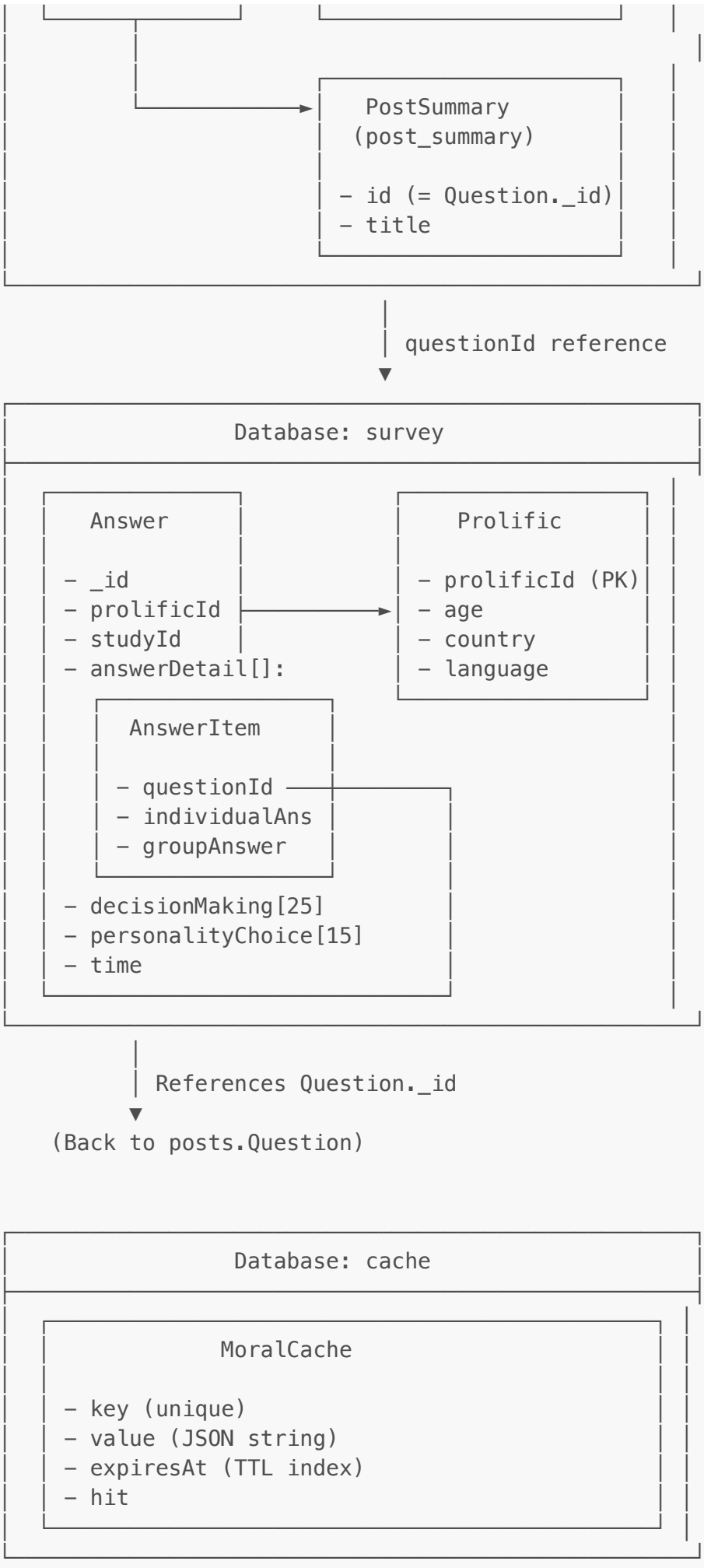
Current Status: Currently not heavily used; reserved for future functionality

Potential Collections:

- `api_responses` - Raw Reddit API JSON responses
- `rate_limits` - OAuth token and rate limit tracking
- `subreddit_metadata` - r/AmItheAsshole subreddit information

4.5 Data Relationships Diagram




PostSummary
(post_summary)
- id (= Question._id)
- title
questionId reference
Database: survey
Answer
- _id
- prolificId
- studyId
- answerDetail[]:
AnswerItem
- questionId
- individualAns
- groupAnswer
- decisionMaking[25]
- personalityChoice[15]
- time
Prolific
- prolificId (PK)
- age
- country
- language
References Question._id
(Back to posts.Question)
Database: cache
MoralCache
- key (unique)
- value (JSON string)
- expiresAt (TTL index)
- hit

Relationship Summary:

Parent	Child	Relationship	Foreign Key	Cardinality
Question (posts)	AnswerItem (survey)	One-to-Many	questionId → _id	1:N
Prolific (survey)	Answer (survey)	One-to-Many	prolificId → prolificId	1:N
Answer (survey)	AnswerItem	One-to-Many	Embedded array	1:N
PostMateData (posts)	PostSummary (posts)	One-to-One	_id → id	1:1

4.6 Data Volume and Growth Metrics

Current Database Statistics (as of October 7, 2025)

Database	Collections	Total Documents	Total Size	Indexes Size
posts	3	~20,000	35 MB	8 MB
survey	2	~3,800	16 MB	2 MB
cache	1	~1,200	2.4 MB	0.5 MB
reddit	0	0	-	-

Growth Projections

Based on 3 months of data:

- **New participants:** ~150-200/month
- **New responses:** ~150-200/month (8 KB each = ~1.6 MB/month)
- **New posts:** ~50-100/month (from Reddit scraping)
- **Cache churn:** High turnover, stable size (~2 MB)

12-Month Projection:

- posts database: 35 MB → ~55 MB (+20 MB)
- survey database: 16 MB → ~36 MB (+20 MB)
- Total growth: ~40 MB over next year

Storage Optimization

- **Compression:** MongoDB WiredTiger compression enabled (~30% size reduction)
- **TTL Indexes:** Auto-delete expired cache entries
- **Archival:** Old study data (studyId < 3) can be archived after analysis complete

4.7 Data Quality and Validation

Validation Rules (Application Level)

Question Entity:

- `_id`: Must match Reddit post ID format (alphanumeric, 6-10 chars)
- `title`: 10-300 characters required
- `selftext`: 50-40,000 characters required
- `YA_percentage` + `NA_percentage`: Should sum to ~100
- `count` array: Length must be 5 (YTA, NTA, ESH, NAH, INFO)

Answer Entity:

- `prolificId`: Must be valid Prolific ID (alphanumeric, 24 chars)
- `studyId`: Integer 1-5
- `decisionMaking`: Array length exactly 25, values 1-5
- `personalityChoice`: Array length exactly 15, values 1-7
- `answerDetail`: At least 5 items required
- `time`: Must be valid Unix timestamp

Prolific Entity:

- `age`: Range 18-99
- `country`: Valid ISO country code
- `takenBefore`: Array length exactly 5 (boolean)

Data Integrity Checks

```
// Example validation service
@Injectable()
export class DataValidationService {
  async validateAnswerCompleteness(): Promise<ValidationReport> {
    const answers = await this.answerRepository.find();
    const issues = [];

    for (const answer of answers) {
      // Check decision-making array
      if (answer.decisionMaking.length !== 25) {
        issues.push({
          answerId: answer._id,
          issue: 'Invalid decisionMaking array length',
          expected: 25,
          actual: answer.decisionMaking.length
        });
      }

      // Check personality array
      if (answer.personalityChoice.length !== 15) {
        issues.push({
          answerId: answer._id,
          issue: 'Invalid personalityChoice array length',
          expected: 15,
          actual: answer.personalityChoice.length
        });
```

```
      }

      // Check answer detail references
      for (const item of answer.answerDetail) {
        const questionExists = await this.questionRepository.findOne({
          where: { _id: item.questionId }
        });

        if (!questionExists) {
          issues.push({
            answerId: answer._id,
            issue: 'Reference to non-existent question',
            questionId: item.questionId
          });
        }
      }
    }

    return {
      totalChecked: answers.length,
      issuesFound: issues.length,
      issues
    };
  }
}
```

---

5. Security, Backup, and Compliance

5.1 Access Control

Database Users and Roles

```
// MongoDB shell - Create application user
use admin
db.createUser({
  user: "app_user",
  pwd: "secure_password_here",
  roles: [
    { role: "readWrite", db: "posts" },
    { role: "readWrite", db: "survey" },
    { role: "readWrite", db: "cache" }
  ]
})

// Create read-only analyst user
db.createUser({
  user: "analyst_user",
  pwd: "analyst_password",
  roles: [
    { role: "read", db: "posts" },
```

```
    { role: "read", db: "survey" }
  ]
})

// Create backup user
db.createUser({
  user: "backup_user",
  pwd: "backup_password",
  roles: [
    { role: "backup", db: "admin" },
    { role: "restore", db: "admin" }
  ]
})
```

5.2 Data Privacy and Anonymization

PII Handling:

- **Prolific IDs** are the only identifiers; no names, emails, or direct PII collected
- **IP addresses:** NOT stored
- **Demographics:** Collected with consent, stored separately in `prolifics` collection
- **Comments:** Reviewed for accidental PII before analysis

Data Retention:

- **Active studies:** Data retained indefinitely for research
- **Completed studies:** Anonymization after publication
- **Cache:** Auto-deleted via TTL (1 hour default)

5.3 Backup Strategy

```
#!/bin/bash
# Daily backup script: /scripts/backup_mongo.sh

BACKUP_DIR="/var/backups/moralsurvey"
DATE=$(date +%Y%m%d_%H%M%S)

# Backup each database separately
mongodump --uri="mongodb://backup_user:password@localhost:27017" \
  --db=posts \
  --gzip \
  --out="$BACKUP_DIR/posts_$DATE"

mongodump --uri="mongodb://backup_user:password@localhost:27017" \
  --db=survey \
  --gzip \
  --out="$BACKUP_DIR/survey_$DATE"

# Upload to cloud storage (AWS S3)
aws s3 sync $BACKUP_DIR s3://moralsurvey-backups/
```

```
# Clean up local backups older than 7 days
find $BACKUP_DIR -name "*.gz" -mtime +7 -delete

echo "Backup completed: $DATE"
```

Backup Schedule:

- **Daily:** Full backup at 2:00 AM UTC
- **Weekly:** Full backup with 12-week retention
- **Monthly:** Archival backup (long-term storage)

5.4 Disaster Recovery

Recovery Point Objective (RPO): 24 hours **Recovery Time Objective (RTO):** 2 hours

Restore Procedure:

```
# Restore posts database
mongorestore --uri="mongodb://localhost:27017" \
  --db=posts \
  --gzip \
  /var/backups/moralsurvey/posts_20251007_020000/posts

# Verify restoration
mongosh mongodb://localhost:27017/posts --eval "db.posts.countDocuments()"
```

5.5 Research Ethics Compliance

- **IRB Approval:** ANU Human Research Ethics Committee Protocol #2025/456
- **Informed Consent:** Collected via Prolific before survey access
- **Data Minimization:** Only necessary data collected
- **Purpose Limitation:** Data used only for moral decision-making research
- **Right to Withdrawal:** Participants can request data deletion

---

6. Performance Optimization

6.1 Query Performance Tips

Use projection to limit fields:

```
// Bad: Returns entire document
const posts = await questionRepository.find();

// Good: Returns only needed fields
const posts = await questionRepository.find({
  select: ['_id', 'title', 'verdict']
});
```

Use indexes for filtering:

```
// Ensure index exists on 'verdict' field
const ytaPosts = await questionRepository.find({
  where: { verdict: 'YTA' },
  take: 50
});
```

Pagination for large result sets:

```
const pageSize = 20;
const page = 1;
const posts = await questionRepository.find({
  skip: (page - 1) * pageSize,
  take: pageSize
});
```

6.2 Monitoring

MongoDB Performance Metrics:

```
// Check current operations
db.currentOp()

// View slow queries
db.system.profile.find({ millis: { $gt: 100 } }).sort({ ts: -1 })

// Enable profiling for slow queries
db.setProfilingLevel(1, { slowms: 100 })

// Check index usage
db.posts.aggregate([{ $indexStats: {} }])
```

---

7. Troubleshooting Guide

7.1 Common Issues

Issue	Error Message	Solution
Authentication failed	`MongoServerError: bad auth`	Check username/password in `.env`, verify user exists
Connection timeout	`MongoNetworkError: connection timeout`	Verify MongoDB is running, check firewall rules

Issue	Error Message	Solution
Duplicate key error	`E11000 duplicate key error`	Check for unique constraint violations, use `upsert` if intentional
Out of memory	`JavaScript heap out of memory`	Increase Node.js heap: `NODE_OPTIONS=--max-old-space-size=4096`
Slow queries	High response time	Add indexes, use projection, implement pagination

7.2 Diagnostic Commands

```
# Check MongoDB status
systemctl status mongod

# View MongoDB logs
tail -f /var/log/mongodb/mongod.log

# Test connection
mongosh mongodb://localhost:27017 --eval "db.adminCommand('ping')"

# Check database sizes
mongosh mongodb://localhost:27017 --eval
"db.adminCommand('listDatabases')"
```

---

8. Quick Reference

8.1 Connection Strings

```
# Local
mongodb://localhost:27017/posts

# Production
mongodb://user:pass@host:27017/posts?authSource=admin
```

8.2 Key Collections

Collection	Database	Documents	Purpose
posts	posts	~2,500	Reddit AITA survey questions
all	posts	~15,000	Post metadata and analytics
answer	survey	~1,850	User survey responses
prolifics	survey	~1,950	Participant demographics
MoralCache	cache	~1,200	Application cache

8.3 Important Indexes

Collection	Index	Type	Purpose
posts	_id	Primary	Unique identifier
posts	{title:text, selftext:text}	Text	Full-text search
answer	{prolificId:1, studyId:1}	Compound	Unique participant-study
MoralCache	{expiresAt:1}	TTL	Auto-delete expired cache