MongoDB Atlas Vector Search Setup – RAG Mongo Workspace (Testleaf)

A concise, step-by-step guide to connect to MongoDB Atlas, create a **Vector Search** index, ingest embeddings, and verify similarity search using the provided RAG Mongo workspace.

---

0) Prerequisites

- **MongoDB Atlas** cluster with **Atlas Search** enabled and org/project access.
- **Database User** with read/write privileges for target DB/collection.
- **Node.js** 18+ and **npm** installed locally.
- **RAG Mongo workspace** codebase (unzipped locally).
- **Embedding provider/API** credentials (as required by your code).
- Ability to edit a local **.env** file.

Security Note: Never share real credentials in documents or chats. If a password contains special characters (e.g., `@`), it **must** be URL-encoded (e.g., `@` → `%40`). Rotate any credentials that may have been exposed.

---

1) Get the Atlas connection string

1. In Atlas, open your **Cluster** → **Connect** → **Drivers**.
2. Click **Copy** to copy the connection string template. It will look like:

```
mongodb+srv://<username>:<password>@<cluster-hostname>/?retryWrites=true&w=majority&appName=<appName>
```

Replace `<username>`, `<password>`, `<cluster-hostname>`, and `<appName>` with your actual values. Ensure the password is URL-encoded.

If you **don’t have credentials**:

- Go to **Database Access** → edit an existing DB user → change/set the password → copy it → update your connection string in your `.env`.

---

2) Identify Database & Collection

From Atlas → **Data Explorer** → connect to your cluster and note:

- **Database name (DB):** `db_test_collection_test_cases_v1cases`
- **Collection name (COLLECTION):** `<your_collection_name>` (replace with your actual collection)

Ensure the DB and collection values here match what you'll put in your `.env`.

---

3) Create the Atlas Vector Search index

1. In Atlas, go to **Atlas Search** → **Create Search Index.**
2. Choose **Vector Search.** Name the index (recommended): `test_cases`.
3. Pick your **Database** and **Collection** (from Step 2).
4. Choose **JSON Editor** → **Next** → delete any prefilled content and paste this schema:

```
{
  "fields": [
    {
      "type": "vector",
      "path": "embedding",
      "numDimensions": 1536,
      "similarity": "cosine"
    },
    {
      "type": "filter",
      "path": "id"
    },
    {
      "type": "filter",
      "path": "module"
    },
    {
      "type": "filter",
      "path": "title"
    },
    {
      "type": "filter",
      "path": "description"
    },
    {
      "type": "filter",
      "path": "expectedResults"
    }
  ]
}
```

5. Click **Create Vector Search Index.**

6. Wait until the index status is **Ready**. A fresh index will show **0 documents** until you ingest data.

Keep names consistent: If you name the index `test_cases`, use the same value in your `.env` (see next section).

---

4) Configure and run the ingestion pipeline

1. **Import** the RAG Mongo workspace (unzipped code) into your local VSCode.
2. Open the root **.env** and set the following (example template):

```
# MongoDB
MONGODB_URI=mongodb+srv://<username>:<password>@<cluster-
hostname>/?retryWrites=true&w=majority&appName=<appName>
DB_NAME=db_test_collection_test_cases_v1cases
COLLECTION_NAME=<your_collection_name>
INDEX_NAME=test_cases

# Embeddings / API (replace with your real values or provider keys)
API_BASE=https://api.testleaf.com
API_EMAIL=<your_email>
API_KEY=<your_api_key>
```

3. Install dependencies and generate data/embeddings:

```
# From repository root
npm install

# (If the workspace has a client app, ensure node_modules exist in both root and
/client)
# cd client && npm install && cd ..

# Convert Excel test cases to JSON
node src/scripts/excel-to-json.js
# Verify data/testcases.json exists and contains ~100 tests

# Create embeddings and store them in MongoDB (also triggers indexing)
node src/scripts/create-embeddings-store.js
# In Atlas, confirm documents appear in your collection and index begins serving
```

Allow time for all ~100 test cases to ingest. In Atlas, you should see the collection populated and the **Vector Search** index serving.

---

5) Run and verify a vector search

1. Open `src/scripts/search-vector-db.js` and update the query string (e.g., **line 25**):

```
const query = "payment failure with expired card"; // example query
```

2. Execute the search script:

```
node src/scripts/search-vector-db.js
```

3. Review the console output. You should see results with a **score** (similarity). Example (shape will vary):

```
Top matches (limit=5):
1) TC-042 – Payment with expired card → score: 0.912
2) TC-017 – Declined transaction (insufficient funds) → score: 0.887
...
```

If you get **no results**, verify DB/collection/index names, that embeddings were created, and your `.env` values.

---

6) Validation checklist

- Atlas connection string in `.env` is correct (password URL-encoded).
- DB/collection match Atlas **Data Explorer** values.
- Atlas **Vector Search** index named `test_cases` (or your chosen name) is **Ready**.
- `data/testcases.json` exists and contains ~100 items.
- After running `create-embeddings-store.js`, the collection shows ~100 docs with an `embedding` array field.
- `search-vector-db.js` returns hits with a similarity **score**.

---

7) Troubleshooting

A. SSL/TLS errors (e.g., `ssl3_read_bytes: tlsv1 alert internal error`)

- Use a stable LTS Node version (≥ 18). Avoid outdated Node versions.
- If behind a corporate proxy with TLS interception, set `NODE_EXTRA_CA_CERTS=/path/to/corporate-rootCA.pem`.
- Ensure your connection string uses `mongodb+srv://` and you're not forcing insecure TLS.

B. Index stuck / not Ready

- Confirm you created a **Vector Search** index on the **correct DB/collection**.

- For large data loads, allow time to build. For ~100 docs this should be quick.

C. No documents after ingestion

- Re-check `.env` values; the script writes to `DB_NAME` / `COLLECTION_NAME`.
- Inspect script logs for API/model errors or rate limits.
- Ensure your embedding model's dimension matches the index schema (1536 vs other).

D. Special characters in password

- URL-encode before placing in the connection string (e.g., `World@1234` → `World%401234`).

E. Query returns irrelevant results

- Try a more specific natural-language query (include domain terms like *payments*, *refund*, *negative flow*).
- Verify that each document contains a good textual representation (title/description/expectedResults) used to build embeddings.

---

8) (Optional) Atlas shell verification

If you prefer to test within Atlas (or MongoDB Shell) using `$vectorSearch` directly, ensure you can produce a `queryVector` (embedding) for your text first. Example pipeline:

```
// Replace <collection> with your collection name
// Replace queryVector with a real 1536-dimensional embedding array

db.<collection>.aggregate([
  {
    $vectorSearch: {
      index: "test_cases",
      path: "embedding",
      queryVector: [/* 1536 floats */],
      numCandidates: 100,
      limit: 5,
      filter: { module: "payments" }
    }
  },
  {
    $project: { title: 1, module: 1, score: { $meta: "vectorSearchScore" } }
  }
]);
```

---

9) Hand-off notes

- Keep index, DB, and collection names **consistent** across Atlas, `.env`, and scripts.
- If you change the embedding **model**, update **both** the index schema (`numDimensions`) and `.env` to match.
- For team onboarding, share this guide plus a sample `.env.example` without secrets.

`.env.example` (shareable template)

```
MONGODB_URI=mongodb+srv://<user>:<pass>@<cluster>/?retryWrites=true&w=majority&app
Name=<appName>
DB_NAME=db_test_collection_test_cases_v1cases
COLLECTION_NAME=<your_collection_name>
INDEX_NAME=test_cases

API_BASE=https://api.testleaf.com
API_EMAIL=<your_email>
API_KEY=<your_api_key>
```
