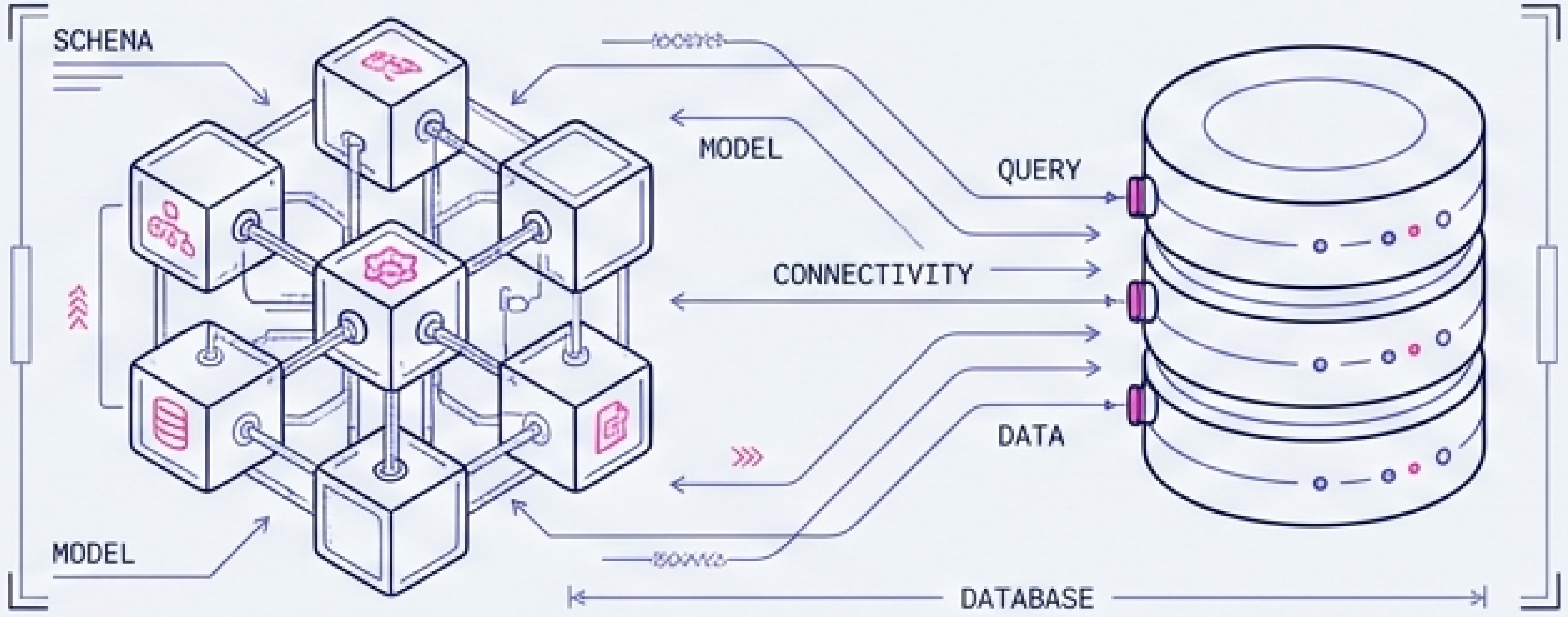
MONGOOSE FUNDAMENTALS

Your ODM for MongoDB

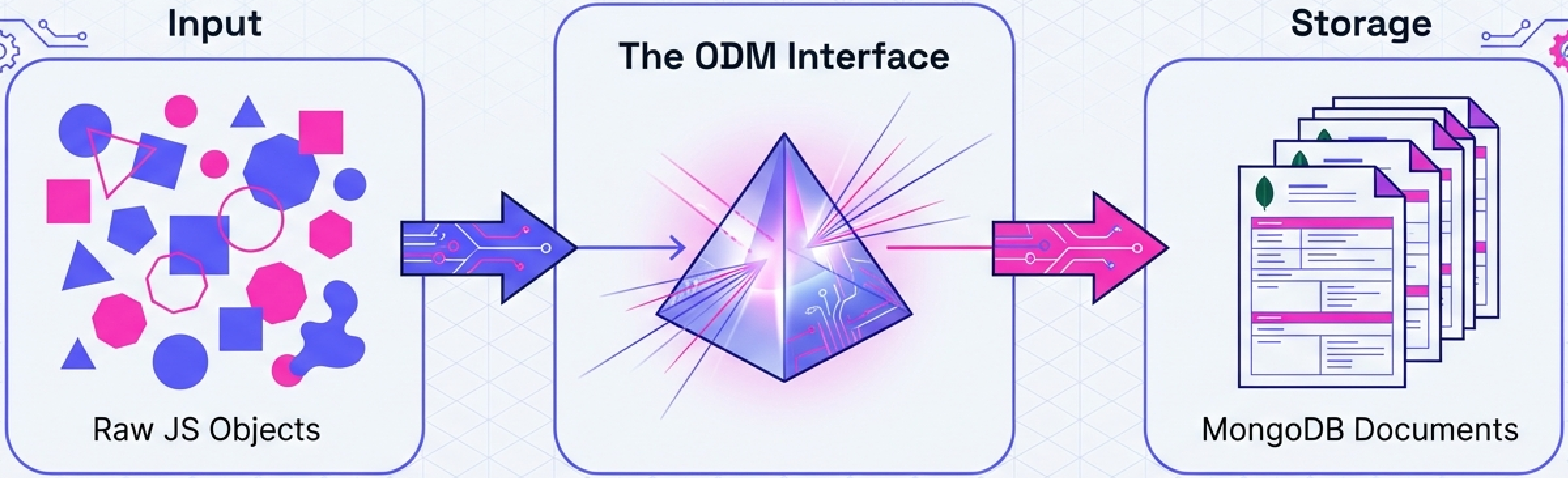


Node2Know • Professor Solo

Module: Database Connectivity

What is an ODM?

The Universal Translator



Object-Document Mapping (ODM) bridges the gap between Node.js code and MongoDB storage. It translates raw JavaScript objects into structured database documents and back again.

Key Insight: Gives you the speed of a document store with the discipline of a schema.

Why Mongoose?

Order out of Chaos

Validation

Prevents saving malformed data. No bad inputs allowed.

Defaults

Automatically fills in the blanks (e.g., dates, flags).

Casting

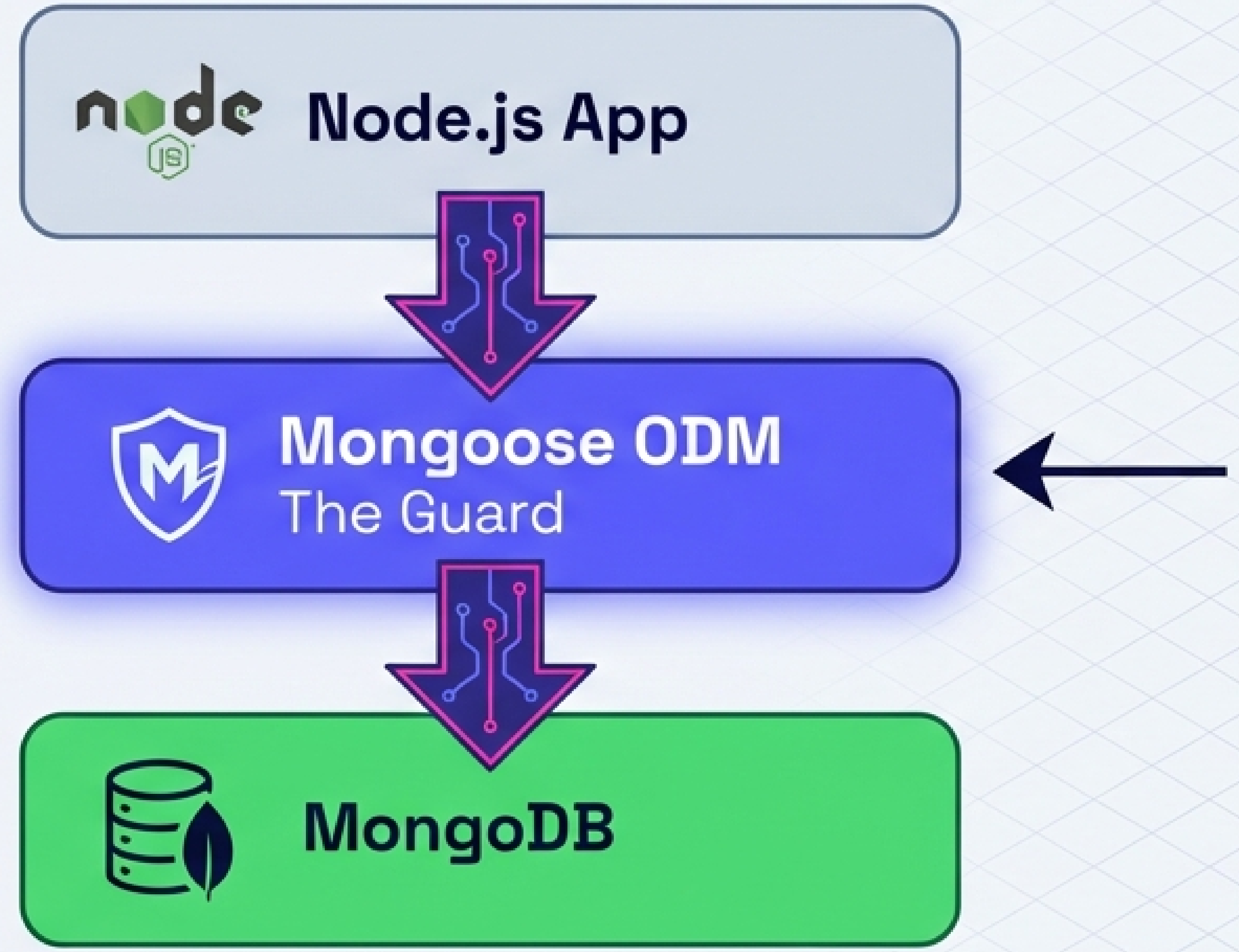
Converts types automatically. String “123” becomes Number 123.

Abstraction

Hides complex, low-level query strings behind clean methods.

The Architecture

Where Mongoose Lives



- Mongoose intercepts data *before* it hits the database.
- It lives inside your Node ☑ application, not the database server.

Establishing the Link

Connect Once, Run Forever

```
const mongoose = require('mongoose');
require('dotenv').config();

// Connect immediately
mongoose.connect(process.env.MONGO_URI);

// Listen for events
const db = mongoose.connection;
db.on('error', console.error);
db.once('open', () => console.log('Connected!'));
```

Connection Pool: Mongoose manages connections in the background.

Security: process.env keeps secrets safe.

Event Listener: db.once('open') is the Green Light.

Graceful Shutdown

Clean Up Your Toys

When the Node process terminates (Ctrl+C), open connections can hang.

We listen for termination signals (SIGINT) to close the shop properly.

```
process.on('SIGINT', async () => {
  console.log('Closing connection...');
  await mongoose.connection.close();
  process.exit(0);
});
```

NotebookLM

The Core Trinity

SCHEMA, MODEL, DOCUMENT

SCHEMA

The Blueprint.
Defines structure, rules, and types.

MODEL

The Factory.
The constructor interface used to query and create.

DOCUMENT

The Instance.
The actual record (data) sitting in the database.

Step 1: The Schema

Defining the Rules

```
const mongoose = require('mongoose');

const projectSchema = new mongoose.Schema({
  title:      { type: String, required: true },
  tagline:    String,
  stack:      [String], // Array of strings
  tags:       [String],
  stars:      { type: Number, default: 0 },
  isFeatured: { type: Boolean, default: false }
});
```

Enforces Data Consistency

Fills Blanks Automatically

NotebookLM

Step 2: The Model

Constructing the Factory

```
// Syntax: .model('Name', schema)
const Project = mongoose.model('Project', projectSchema);

module.exports = Project;
```

Transformation

Project
(Singular, Uppercase)

Mongoose Magic

projects
(Plural, Lowercase)

Naming Rule: Mongoose automatically looks for the **plural**, **lowercase** version of your model name as the collection.

Step 3: Create

Writing to the Database

```
// 1. Create a document instance in memory
const newProject = new Project({
  title: 'Node2Know App',
  stack: ['Node', 'Express', 'Mongoose'],
  isFeatured: true
});

// 2. Persist to MongoDB (Async!)
await newProject.save();
```

new Project(): Creates a transient object (RAM only).

.save(): Writes the object to the disk.

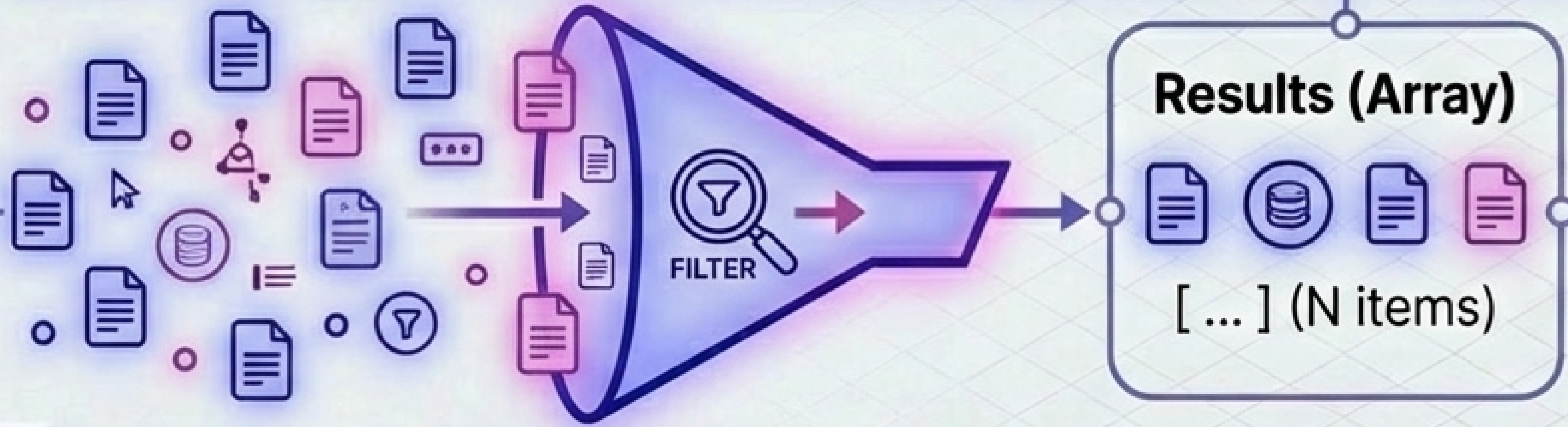
CRITICAL: Always use 'await'. Database operations take time.

Step 4: Read (Find All)

Fetching the Gallery

```
// Find all projects that are featured
const results = await Project.find({ isFeatured: true });

// Returns an Array (empty [] if none found)
if (results.length > 0) {
  console.log(`Found ${results.length} projects.`);
}
```



Pass **{ key: value }** to narrow results.

Pass **{}** to return everything.

Precision Finding

findById vs findOne

System Lookup

```
const p1 = await Project.
  findById('65c123...');
```

The fastest way to look up a unique system record.

User Lookup

```
const p2 = await Project.findOne({
  title: 'Node2Know App'
});
```

Great for user-friendly URLs or searching by unique titles.

{ } Both return a **SINGLE OBJECT** (or null), not an Array.

Flexible Search

Using $or and $regex

```
const search = 'Node';

// Find projects with 'Node' in Title OR Tagline
const results = await Project.find({
  $or: [
    { title:   { $regex: search, $options: 'i' } },
    { tagline: { $regex: search, $options: 'i' } }
  ]
});
```

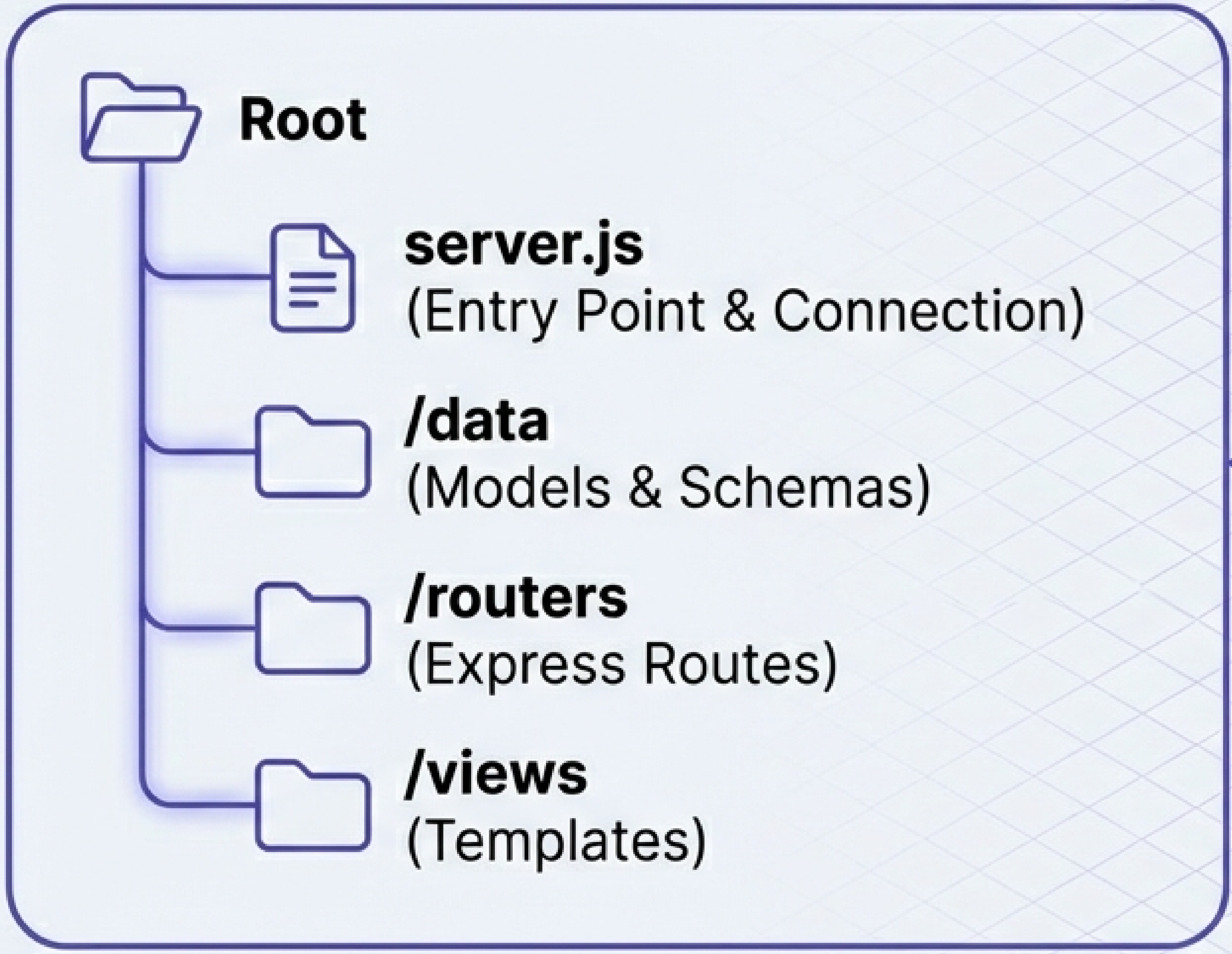
$or: Matches if ANY condition is true.

$regex: Partial matching (Fuzzy search).

$options 'i': Case-insensitive.

Project Structure

Keeping the Lab Clean



Root

- **server.js** (Entry Point & Connection)
- **/data** (Models & Schemas)
- **/routers** (Express Routes)
- **/views** (Templates)

Define Schemas in **/data.**

Handle logic in **/routers.**

Don’t clutter **server.js** with model definitions.

The Pocket Mong-o-Dex

Quick Reference

`new Schema({...})`	Define Structure
`model('Name', schema)`	Create Factory
`new Model().save()`	Create & Persist
`Model.find(filter)`	Get Array
`Model.findOne(filter)`	Get Single Object

p.s., keep learning!