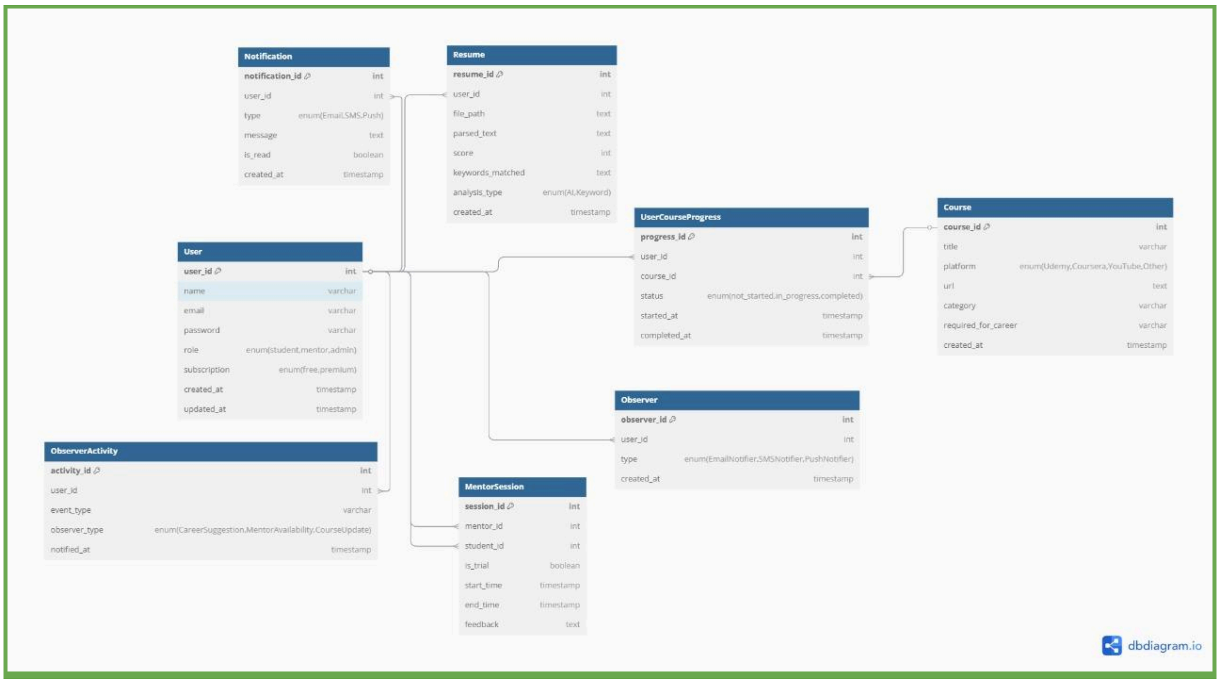
First: ER diagram



Second: Database schema

1. User Collection (`users`)

- **Description:**

Stores all users in the system, including students, mentors, and admins. It uses **single collection inheritance** with a `role` discriminator.

- **Fields:**

Field	Type	Required	Description
ID	ObjectId	yes	Auto-generated MongoDB ID
name	string	yes	Full name of the user
email	string	yes	Must be unique
password	string	yes	Hashed password
subscription	string	no	Enum: `'free'`, `'premium'`; default `'free'`

role	string	yes	Enum: `'student'`, `'mentor'`, `'admin'`; discriminator field
observer	[ObjectId]	no	Array of `Observer` references
createdAt	Date	yes	Timestamp (auto)
updatedAt	Date	yes	Timestamp (auto)

Notes:

- Uses `discriminatorKey: 'role'` to support different role logic.
- Sub-roles (`Student`, `Mentor`) may be extended with Mongoose discriminators.

---

2. Resume Collection (`resumes`)

- **Description:**

  Stores uploaded resumes and their analysis results for students.

- **Fields:**

Field	Type	Required	Description
ID	ObjectId	yes	Auto-generated MongoDB ID
user	ObjectId	yes	Reference to the User who uploaded
filePath	string	yes	Location of the uploaded file
text	string	yes	Extracted text from the resume
score	Number	yes	Resume quality score
keywordsMatched	[string]	yes	Skills or keywords identified

createdAt	Date	yes	Auto timestamp

---

3. Observer Collection (`observers`)

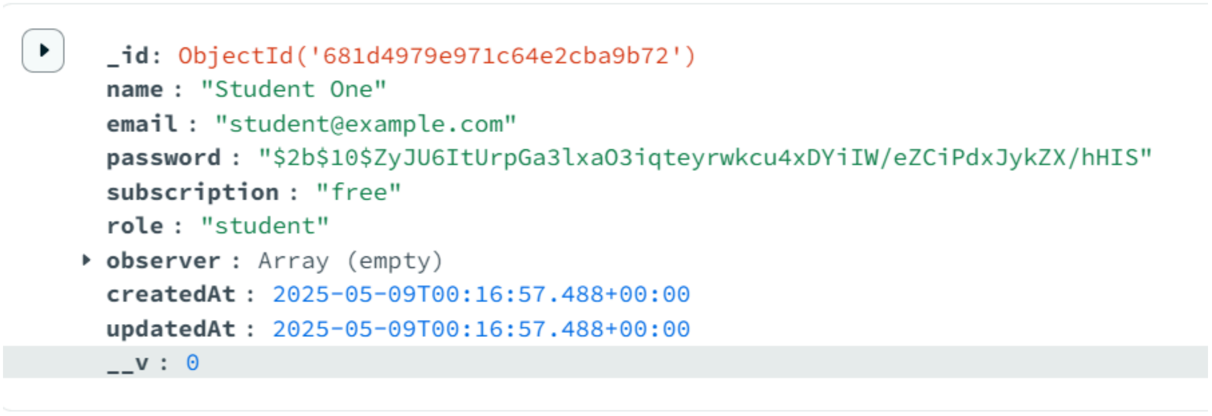
- **Description:**

  Stores observers for the Observer pattern (e.g., email notifications).

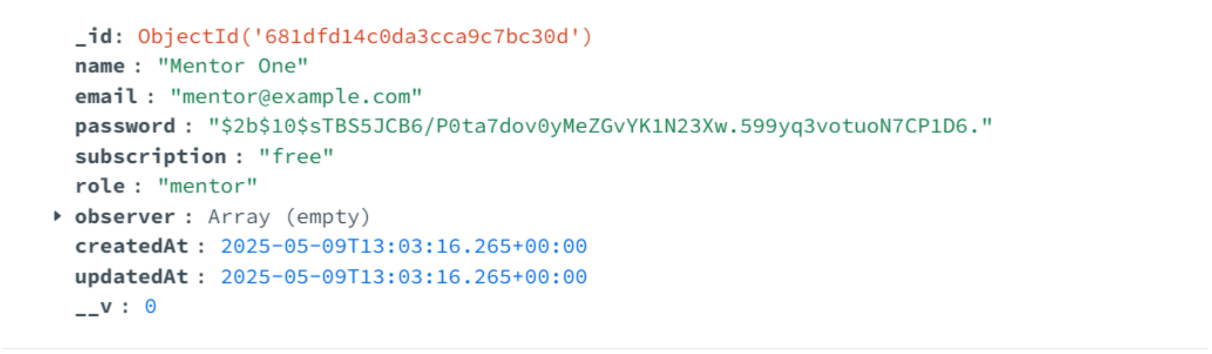
- **Fields:**

Field	Type	Required	Description
ID	ObjectId	yes	Auto-generated ID
Type	string	yes	Type of observer (e.g., EmailNotifier)
User	ObjectId	yes	User being observed (optional)

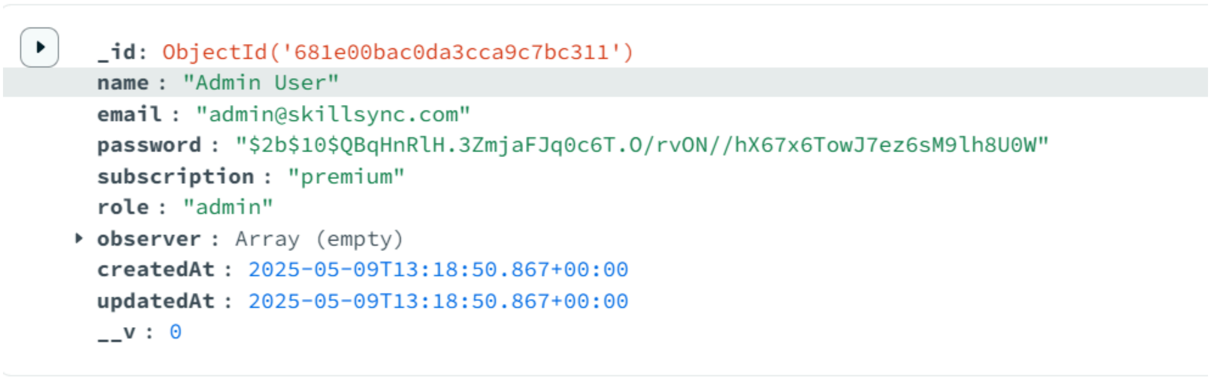
- **Screenshot from MongoDB accessing SkillSync:**



```
_id: ObjectId('681d4979e971c64e2cba9b72')
name : "Student One"
email : "student@example.com"
password : "$2b$10$ZyJU6ItUrpGa3lxaO3iqteyrwkcu4xDYiIW/eZCiPdxJykZX/hHIS"
subscription : "free"
role : "student"
observer : Array (empty)
createdAt : 2025-05-09T00:16:57.488+00:00
updatedAt : 2025-05-09T00:16:57.488+00:00
__v : 0
```



```
_id: ObjectId('681dfd14c0da3cca9c7bc30d')
name : "Mentor One"
email : "mentor@example.com"
password : "$2b$10$sTBS5JCB6/P0ta7dov0yMeZGvYK1N23Xw.599yq3votuoN7CP1D6."
subscription : "free"
role : "mentor"
observer : Array (empty)
createdAt : 2025-05-09T13:03:16.265+00:00
updatedAt : 2025-05-09T13:03:16.265+00:00
__v : 0
```



```
_id: ObjectId('681e00bac0da3cca9c7bc311')
name : "Admin User"
email : "admin@skillsync.com"
password : "$2b$10$QBqHnRlH.3ZmjaFJq0c6T.O/rvON//hX67x6TowJ7ez6sM9lh8U0W"
subscription : "premium"
role : "admin"
observer : Array (empty)
createdAt : 2025-05-09T13:18:50.867+00:00
updatedAt : 2025-05-09T13:18:50.867+00:00
__v : 0
```

Third: CRUD Operations implementation

- **Overview:**

This section describes how the core entities in the SkillSync system perform **Create, Read, Update, and Delete (CRUD)** operations through RESTful APIs built with Express and MongoDB using Mongoose ODM.

---

1. User Management (using `/api/auth` and `/api/admin` routes)

Operation	API Endpoint	HTTP Method	Access	Description
Create	`/api/auth/register`	POST	Public	Registers a new user. Validates email uniqueness, hashes password, and saves user to the database
Read (self)	`/api/auth/login`	POST	Public	Authenticates users with email and password. Returns JWT token and user details.
Read (admin)	`/api/admin/users`	GET	Admin only	Retrieves a list of all users (excluding passwords).
Read (by ID)	`/api/admin/users/:id`	GET	Admin only	Retrieves a single user by ID. Useful for viewing

				profile data. *(optional: can be added)*
Update	`/api/users/:id`	PUT	Authenticated users	Allows users to update their own profile. *(optional: can be implemented)*
Delete	`/api/admin/users/:id`	DELETE	Admin only	Deletes a user by ID from the system. Performs role check before deletion.

Model Used: `User`
Validation: Enforced using Mongoose schema (e.g., `required`, `unique`, `enum`)
Security: Passwords are hashed using `bcrypt`, JWT is used for authentication.

---

2. Resume Upload & Analysis (`/api/resume`)

Operation	API Endpoint	HTTP Method	Access	Description
Create	`/api/resume/upload`	POST	Student only	Uploads a resume (PDF/DOCX), extracts text, analyzes it using selected analyzer, and returns a score/keywords
Read	*(to be implemented)*	GET	Student/Admin	Could allow users to view their previous uploads or analysis history

Update	*(not required)*	—	—	Resume updates not supported; considered immutable
Delete	*(optional)*	DELETE	Student/Admin	Could be implemented to allow removal of uploaded resumes.

Model Used: `Resume`
Upload Tool: `multer` (for file uploads)
Analysis Tool: Factory pattern selects analyzer based on user subscription level.
Notification: Observer pattern triggers notifications after upload.

---

3. Notification System (Observer Pattern)

This module is not accessed via CRUD endpoints but functions internally as part of application logic.

- **Create**: When a resume is analyzed, an `EmailNotifier` instance is attached to the `NotificationService`, which triggers a notification.
- **Read/Update/Delete**: Not stored in DB (for now). If extended, could be represented in an `observers` collection.

Technologies & Patterns Used:

Aspect	Tool/Pattern	Purpose
Persistence	MongoDB + Mongoose	Document storage and schema definition
Relationships	Mongoose `.populate()`	Link resumes to users and observers (if implemented)
Authentication	JWT + bcrypt	Secure login and route protection
Authorization	Middleware	Role-based access to protected routes
Upload & Parsing	multer, pdf-parse, mammoth	File handling and content extraction
Business Logic	Service Layer + Factory Pattern	Modular, scalable backend logic
Notification	Observer Pattern	Sends dynamic updates when events occur

Forth: ORM strategy (if used)

In SkillSync, we use **Mongoose**, which is technically an **ODM** (Object Document Mapper) tailored for MongoDB — the NoSQL database used in this project. Like traditional ORMs (e.g., Sequelize for SQL), Mongoose simplifies data handling by allowing you to work with JavaScript objects instead of raw database queries.

Mongoose was selected because:

- It integrates seamlessly with **MongoDB**
- Supports **schema definitions** for otherwise schema-less MongoDB
- Offers powerful features like:
  - Schema validation
  - Middleware hooks

    - Virtuals and statics
    - Discriminators (used for roles)
- ❖ Well-documented, widely adopted, and highly maintainable for team-based development

---

How Mongoose Is Used in SkillSync

1. Schema Definition

- Each collection (User, Resume) is defined with a Mongoose `Schema`, specifying field types, validations, defaults, and relationships.

2. Model Instantiation & CRUD

- Models like `User` and `Resume` are used for direct interaction with the database
- These replace the need for raw MongoDB queries (db.users.find(...), etc.)

3. Discriminators for Roles

- Using `discriminatorKey`, Mongoose supports inheritance-like behavior. For example, students, mentors, and admins can have different logic but share the same base `User` model.

Summary:

Mongoose was a crucial tool in implementing SkillSync's backend:

- It ensured a structured, safe interface to MongoDB
- Simplified CRUD logic and boosted productivity
- Supported advanced features like role-based inheritance and pre/post middleware
- Enabled better error handling and testability

This ODM approach aligns with the principles of scalable and maintainable backend development.