CSE349 - Advanced Database Systems Design

Major Task - Phase 2

Team 19 – EduVerse

Education – Community

12/12/2025

Presented by:
Ahmed Mohamed Fahmy - 23P0303
Ahmed Mohamed Naguib - 23P0305
Hady Mostafa Abdelaziz - 2301128
Peter Maged Shokry - 23P0192
John George Mikhael - 23P0266
Yousef Amr Said - 23P0288
Zeyad Tamer Darwish Ghoneem - 23P0258

Presented to Eng. Esraa Karam

TABLE OF CONTENT

1.0 PROJECT OVERVIEW

EduVerse is a full-stack educational social platform designed to enhance academic collaboration between students and instructors. Built using the MERN stack (MongoDB, Express.js, React.js, Node.js), the platform provides a comprehensive environment for course management, academic discussions, and real-time communication. This submission represents Phase 2 of the project, a fully functional application with complete frontend-backend integration. Building upon the database schema and design from Phase 1, this phase delivers:

Core Features:

- User Authentication: Secure registration and login with JWT-based sessions, supporting both student and instructor roles.
- Course Management: Course creation, enrolment tracking, and capacity management with instructor assignments.
- Discussion Forum: Post creation with three types (announcements, questions, discussions), threaded comments, and a reaction system (like, love, laugh, shocked, sad).
- Real-time Messaging: Private chat functionality between users with message history and file attachments.
- User Profiles: Customizable profiles with profile pictures, activity tracking, and role-based views.

Technical Implementation:

- Frontend: React.js with React Router for navigation, Axios for API communication, and a responsive dark/light theme UI.
- Backend: Express.js REST API with modular route and service architecture, JWT middleware for authentication.
- Database: MongoDB with Mongoose ODM, featuring 8 collections (Users, Posts, Comments, Reactions, Courses, Chats, Messages, Files) with proper ObjectId references and embedded documents for denormalized data.

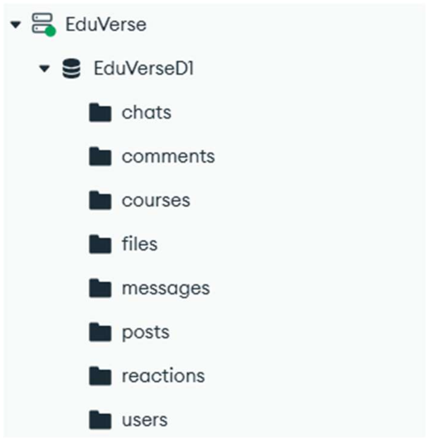
Advanced Database Features:

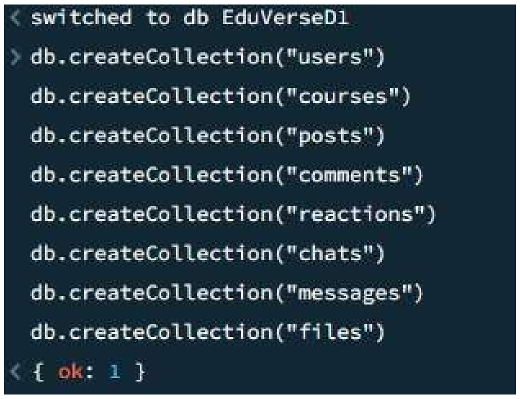
- Aggregation pipelines for analytics and reporting (course engagement, user leaderboards, platform activity metrics and instructor course performance).
- Indexed queries for optimized performance.
- Data validation at both schema and application levels.

The application demonstrates practical implementation of NoSQL database concepts including document embedding, referencing, aggregation frameworks, and schema design patterns suitable for a social-educational platform.

2.0 MONGODB SCRIPT REQUIREMENTS

2.1 Database & Collection Creation





```
< switched to db EduVerseD1
> db.createCollection("users")
  db.createCollection("courses")
  db.createCollection("posts")
  db.createCollection("comments")
  db.createCollection("reactions")
  db.createCollection("chats")
  db.createCollection("messages")
  db.createCollection("files")
< { ok: 1 }
```

2.2 Collections Overview & Database Schema

Collection	Purpose	Key Relationships
Users	Store user accounts and profiles	Referenced by Posts, Comments, Reactions, Messages, Chats
Courses	Manage academic courses	References Users (instructors), Referenced by Posts
Posts	Discussion forum content	References Users, Courses, Files
Comments	Responses to posts	References Users, Posts
Reactions	Emoji reactions on posts	References Users, Posts
Chats	Conversation threads	References Users
Messages	Individual chat messages	References Users, Chats, Files
Files	Uploaded file storage	Referenced by Posts, Messages

2.2.1 users collection

Stores all user accounts including students and instructors.

```
{
  _id: ObjectId,                      // Auto-generated unique identifier
  name: String,                       // User's display name
  email: String,                      // Unique email address (required)
  password: String,                   // Hashed password (required)
  resetPasswordToken: String,         // Token for password reset (nullable)
  resetPasswordExpires: Date,         // Expiry time for reset token (nullable)
  image: Object,                      // Profile picture metadata { fileId: 
ObjectId }
  level: String,                      // Academic level (e.g., "Freshman", 
"Senior", "Professor")
  courses: [String],                  // Array of enrolled course IDs
  role: String,                       // "student" or "instructor" (default: 
"student")
  createdAt: Date                     // Account creation timestamp
}
```

Design Decisions:

- **image** is stored as an embedded object containing a reference to the Files collection, allowing flexible metadata storage.
- **courses** uses an array of course IDs for quick enrollment lookups.
- **role** enum restricts values to maintain data integrity.
- Password reset fields are nullable to avoid unnecessary storage.

Relationships:

- Referenced by **Courses** via **instructorId.**
- Referenced by **Posts** via **sender.id.**
- Referenced by **Comments** via **sender.id.**
- Referenced by **Reactions** via **senderId.**
- Referenced by **Chats** via **user1.id** and **user2.id.**
- Referenced by **Messages** via **senderId** and **receiverId.**
- References **Files** via **image.fileId**.
- References **Courses** via **courses[] array**.

Indexes: Unique index on **email** for fast lookups and duplicate prevention.

2.2.2 courses collection

Manages academic courses offered on the platform.

```
{
  _id: String,                        // Custom course code (e.g., "CS101",
"DB201")
  name: String,                       // Full course name
  creditHours: Number,                // Course credit hours
  description: String,                // Course description
  instructorId: [ObjectId],           // Array of instructor User IDs (ref: Users)
  enrolled: Number,                   // Current enrollment count (default: 0)
  capacity: Number                    // Maximum capacity (default: 80)
}
```

Design Decisions:

- **_id** uses a custom String (course code) instead of ObjectId for human-readable references.
- **instructorId** is an array to support co-teaching scenarios.
- **enrolled** is denormalized for quick capacity checks without counting enrolled users.

Relationships:

- References Users collection via **instructorId.**
- Referenced by Posts via **coursed.**

2.2.3 posts collection

Stores all discussion forum posts including announcements, questions, and discussions.

```
{
  _id: ObjectId,                      // Auto-generated unique identifier
  sender: {                           // Embedded sender information
    id: ObjectId,                     // Reference to Users collection
    name: String,                     // Denormalized user name
    image: Object                     // Denormalized profile picture
  },
  courseId: String,                   // Reference to Courses collection
  title: String,                      // Post title
  body: String,                       // Post content
  attachmentsId: [ObjectId],          // Array of File IDs (ref: Files)
  type: String,                       // "question", "announcement", or
"discussion"
  deadline: Date,                     // Optional deadline for announcements
  createdAt: Date                     // Post creation timestamp
}
```

Design Decisions:

- **sender** uses embedded document pattern for denormalization, reduces joins when displaying posts.
- **type** enum categorizes posts for filtering and display purposes.
- **attachmentsId** array supports multiple file attachments.
- **deadline** is optional, primarily used for announcement-type posts.

Relationships:

- Embeds partial **Users** data in **sender**.
- References **Courses** via **courseId**.
- References **Files** via **attachmentsId**.
- Referenced by **Comments** and **Reactions**.

2.2.4 comments collection

Stores user comments/replies on posts.

```
{
  _id: ObjectId,                     // Auto-generated unique identifier
  postId: ObjectId,                  // Reference to Posts collection
  sender: {                          // Embedded sender information
    id: ObjectId,                    // Reference to Users collection
    name: String,                    // Denormalized user name
    image: Object                    // Denormalized profile picture
  },
  body: String,                      // Comment content
  createdAt: Date                    // Comment creation timestamp
}
```

Design Decisions:

- Follows same **sender** embedding pattern as Posts for consistency.
- **postId** enables efficient querying of all comments for a specific post.
- Flat structure (no nested replies) simplifies queries and UI rendering.

Relationships:

- References **Posts** via **postId.**
- Embeds partial **Users** data in **sender.**

2.2.5 reactions collection

Stores emoji reactions on posts (like, love, laugh, shocked, sad).

```
{
  _id: ObjectId,                    // Auto-generated unique identifier
  postId: ObjectId,                 // Reference to Posts collection
  senderId: ObjectId,               // Reference to Users collection
  type: String,                     // "like", "love", "shocked", "laugh", or
"sad"
  createdAt: Date                   // Reaction timestamp
}
```

Design Decisions:

- Separate collection (vs. embedded in Posts) allows efficient aggregation and prevents document bloat.
- **type** enum restricts to predefined reaction types.
- One reaction per user per post enforced at application level.

Relationships:

- References **Posts** via **postId.**
- References **Users** via **senderId.**

Common Queries:

- Count reactions by type for a post.
- Check if user has reacted to a post.
- Get reaction summary using aggregation.

2.2.6 chats collection

Represents conversation threads between two users.

```
{
  _id: ObjectId,                    // Auto-generated unique identifier
  user1: {                          // First participant (embedded)
    id: ObjectId,                   // Reference to Users collection
    name: String,                   // Denormalized user name
    image: Object                   // Denormalized profile picture
  },
  user2: {                          // Second participant (embedded)
    id: ObjectId,                   // Reference to Users collection
    name: String,                   // Denormalized user name
    image: Object                   // Denormalized profile picture
  },
  lastMessage: String,              // Preview of most recent message
  updatedAt: Date                   // Last activity timestamp
}
```

Design Decisions:

- Embeds both user details for efficient chat list rendering without joins.
- **lastMessage** denormalized for chat preview display.
- **updatedAt** enables sorting chats by recent activity.
- Two-user structure (not group chat) simplifies querying.

Relationships:

- Embeds partial **Users** data for both participants.
- Referenced by **Messages** implicitly via user IDs.

2.2.7 messages collection

Stores individual messages within chat conversations.

```
{
  _id: ObjectId,                    // Auto-generated unique identifier
  senderId: ObjectId,               // Reference to Users collection (sender)
  receiverId: ObjectId,             // Reference to Users collection (recipient)
  text: String,                     // Message content
  attachmentsId: [ObjectId],        // Array of File IDs (ref: Files)
  replyTo: ObjectId,                // Reference to another Message (nullable)
  createdAt: Date                   // Message timestamp
}
```

Design Decisions:

- Direct **senderId/receiverId** references (not embedded) since messages are queried in bulk.
- **attachmentsId** array supports file sharing in chat.
- **replyTo** enables reply-to-message functionality.
- Messages linked to chats via sender/receiver pair matching.

Relationships:

- References **Users** via **senderId** and **receiverId**.
- References **Files** via **attachmentsId**.
- Self-references via **replyTo** for reply chains.

2.2.8 files collection

Stores uploaded files (images, PDFs, documents) as binary data.

```
{
  _id: ObjectId,                    // Auto-generated unique identifier
  fileName: String,                 // Original file name
  fileType: String,                 // "image", "pdf", or "word"
  fileData: Buffer,                 // Binary file content (required)
  fileSize: Number,                 // File size in bytes
  courseId: ObjectId,               // Optional reference to Courses
  createdAt: Date                   // Upload timestamp
}
```

Design Decisions:

- Binary storage in MongoDB (vs. filesystem) for simplicity and atomic operations.
- **fileType** enum restricts to supported formats.
- **fileSize** stored for quick size checks without reading binary data.
- **courseId** optional for course-specific materials.

Relationships:

- Referenced by **Posts** via **attachmentsId.**
- Referenced by **Messages** via **attachmentsId.**
- Referenced by **Users** via **image.fileId.**

2.3 Schema Design Patterns Used

2.3.1 embedded documents

Used in **Posts**, **Comments**, and **Chats** for sender/user information. This pattern:

- Reduces read-time joins for frequently accessed data
- Trades storage space for query performance
- Requires application-level updates when user data changes

2.3.2 references documents

Used for **Reactions**, **Messages**, and **file attachments**. This pattern:

- Prevents document bloat for one-to-many relationships.
- Enables efficient aggregation queries.
- Maintains data consistency for frequently updated fields.

2.4 CRUD Operations

2.4.1 insert operations

Ex. Create user (registration).

MongoDB Script:

```
db.users.insertOne({
  name: "Ahmed Hassan",
  email: "ahmed@university.edu",
  password: "$2b$10$hashedpasswordhere",  // bcrypt hashed
  level: "Senior",
  role: "student",
  courses: [],
  image: {},
  createdAt: new Date()
})
```

Mongoose Implementation: backend/services/auth.js

```
const user = new User({
  name,
  email,
  password: hashedPassword,
  level: level || "",
  role: role || "student",
});
await user.save();
```

2.4.2 read queries

Ex. Get Comments for Post.

MongoDB Script:

```
db.comments.find({ postId: ObjectId("507f1f77bcf86cd799439022") })
  .sort({ createdAt: 1 })
```

Mongoose Implementation: backend/services/comment.js

```
const comments = await Comment.find({ postId }).sort({ createdAt: 1 }).lean();
```

2.4.3 update queries

Ex. Update Password (Reset)

MongoDB Script:

```
db.users.updateOne(
  { _id: ObjectId("507f1f77bcf86cd799439011") },
  {
    $set: { password: "$2b$10$newhashedpassword" },
    $unset: { resetPasswordToken: "", resetPasswordExpires: "" }
  }
)
```

Mongoose Implementation: backend/services/auth.js

```
user.password = await bcrypt.hash(password, 10);
user.resetPasswordToken = undefined;
user.resetPasswordExpires = undefined;
await user.save();
```

2.4.4 delete queries

Ex. Update Password (Reset)

MongoDB Script:

```
db.users.deleteOne({ _id: ObjectId("507f1f77bcf86cd799439011") })
```

Mongoose Implementation: backend/services/user.js

```
await User.findByIdAndDelete(id);
```

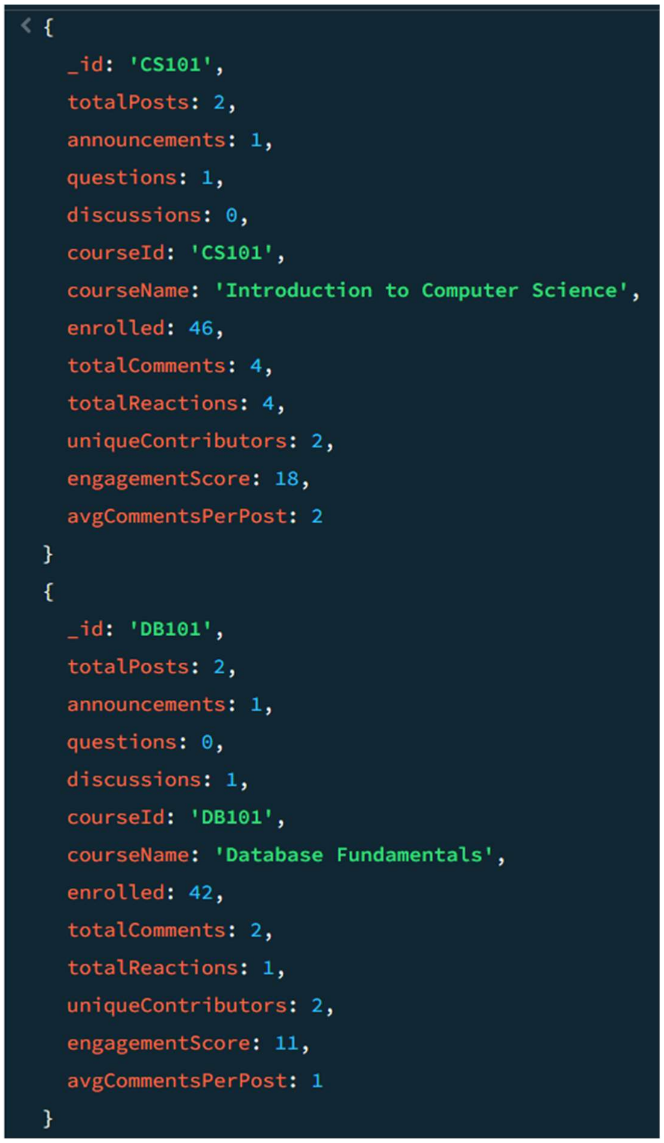
2.5 Aggregation Pipelines

2.5.1 pipeline 1: course engagement analytics

Collection: posts.
Purpose: Analyse engagement metrics per course (posts, comments, reactions).

```
db.posts.aggregate([
  {
    $group: {
      _id: "$courseId",
      totalPosts: { $sum: 1 },
      announcements: {
        $sum: { $cond: [{ $eq: ["$type", "announcement"] }, 1, 0] }
      },
      questions: {
        $sum: { $cond: [{ $eq: ["$type", "question"] }, 1, 0] }
      },
      discussions: {
        $sum: { $cond: [{ $eq: ["$type", "discussion"] }, 1, 0] }
      },
      postIds: { $push: "$_id" },
      uniqueContributors: { $addToSet: "$sender.id" }
    }
  },
  {
    $lookup: {
      from: "comments",
      localField: "postIds",
      foreignField: "postId",
      as: "comments"
    }
  },
  {
    $lookup: {
      from: "reactions",
      localField: "postIds",
      foreignField: "postId",
      as: "reactions"
    }
  },
{
    $lookup: {
      from: "courses",
      localField: "_id",
      foreignField: "_id",
      as: "courseInfo"
    }
  },
  {
    $unwind: {
      path: "$courseInfo",
      preserveNullAndEmptyArrays: true
    }
  },
  {
    $project: {
      courseId: "$_id",
      courseName: { $ifNull: ["$courseInfo.name", "General/Unknown"] },
      enrolled: { $ifNull: ["$courseInfo.enrolled", 0] },
      totalPosts: 1,
      announcements: 1,
      questions: 1,
      discussions: 1,
      totalComments: { $size: "$comments" },
      totalReactions: { $size: "$reactions" },
      uniqueContributors: { $size: "$uniqueContributors" },
      engagementScore: {
        $add: [
          { $multiply: ["$totalPosts", 3] },
          { $multiply: [{ $size: "$comments" }, 2] },
          { $size: "$reactions" }
        ]
      },
      avgCommentsPerPost: {
        $cond: [
          { $eq: ["$totalPosts", 0] },
          0,
          { $round: [{ $divide: [{ $size: "$comments" }, "$totalPosts"] }, 2]
}
        ]
      }
    }
  },
  {
    $sort: { engagementScore: -1 }
  }
])
```

Output:



```
< {
    _id: 'CS101',
    totalPosts: 2,
    announcements: 1,
    questions: 1,
    discussions: 0,
    courseId: 'CS101',
    courseName: 'Introduction to Computer Science',
    enrolled: 46,
    totalComments: 4,
    totalReactions: 4,
    uniqueContributors: 2,
    engagementScore: 18,
    avgCommentsPerPost: 2
  }
  {
    _id: 'DB101',
    totalPosts: 2,
    announcements: 1,
    questions: 0,
    discussions: 1,
    courseId: 'DB101',
    courseName: 'Database Fundamentals',
    enrolled: 42,
    totalComments: 2,
    totalReactions: 1,
    uniqueContributors: 2,
    engagementScore: 11,
    avgCommentsPerPost: 1
  }
```

Note: This screenshot does not contain all the documents, just samples.

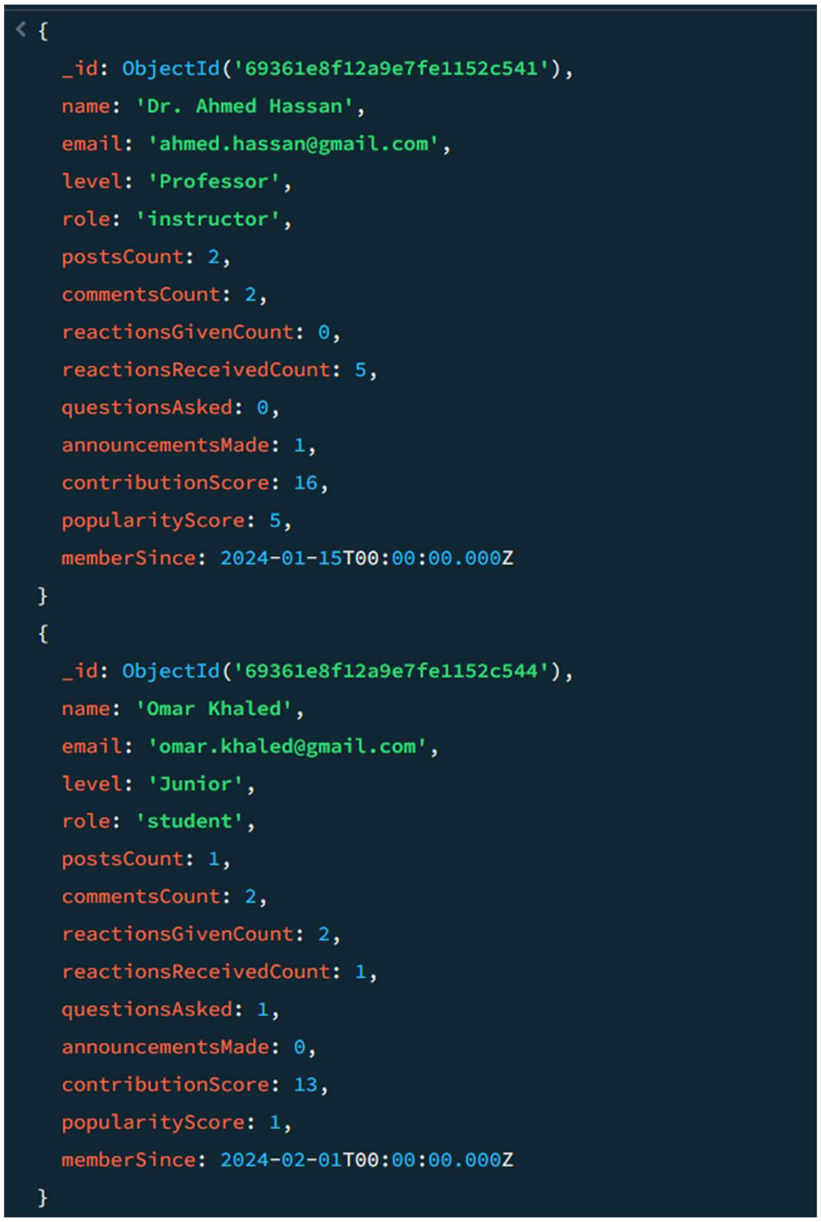
2.5.2 pipeline 2: top contributors leaderboard

Collection: users.

Purpose: Rank users by their contributions (posts, comments, reactions).

```
db.users.aggregate([
  {
    $lookup: {
      from: "posts",
      localField: "_id",
      foreignField: "sender.id",
      as: "posts"
    }
  },
  {
    $lookup: {
      from: "comments",
      localField: "_id",
      foreignField: "sender.id",
      as: "comments"
    }
  },
  {
    $lookup: {
      from: "reactions",
      localField: "_id",
      foreignField: "senderId",
      as: "reactionsGiven"
    }
  },
  {
    $lookup: {
      from: "reactions",
      let: { userPostIds: "$posts._id" },
      pipeline: [
        {
          $match: {
            $expr: { $in: ["$postId", "$$userPostIds"] }
          }
        }
      ],
      as: "reactionsReceived"
    }
  },
  {
    $project: {
      name: 1,
      email: 1,
      role: 1,
      level: 1,
      postsCount: { $size: "$posts" },
      commentsCount: { $size: "$comments" },
      reactionsGivenCount: { $size: "$reactionsGiven" },
      reactionsReceivedCount: { $size: "$reactionsReceived" },
      questionsAsked: {
        $size: {
          $filter: {
            input: "$posts",
            as: "post",
            cond: { $eq: ["$$post.type", "question"] }
          }
        }
      },
      announcementsMade: {
        $size: {
          $filter: {
            input: "$posts",
            as: "post",
            cond: { $eq: ["$$post.type", "announcement"] }
          }
        }
      },
      contributionScore: {
        $add: [
          { $multiply: [{ $size: "$posts" }, 5] },
          { $multiply: [{ $size: "$comments" }, 3] },
          { $size: "$reactionsGiven" }
        ]
      },
      popularityScore: { $size: "$reactionsReceived" },
      memberSince: "$createdAt"
    }
  },
  {
    $sort: { contributionScore: -1 }
  },
  {
    $limit: 10
  }
])
```

Output:



```
< {
    _id: ObjectId('69361e8f12a9e7fe1152c541'),
    name: 'Dr. Ahmed Hassan',
    email: 'ahmed.hassan@gmail.com',
    level: 'Professor',
    role: 'instructor',
    postsCount: 2,
    commentsCount: 2,
    reactionsGivenCount: 0,
    reactionsReceivedCount: 5,
    questionsAsked: 0,
    announcementsMade: 1,
    contributionScore: 16,
    popularityScore: 5,
    memberSince: 2024-01-15T00:00:00.000Z
  }
  {
    _id: ObjectId('69361e8f12a9e7fe1152c544'),
    name: 'Omar Khaled',
    email: 'omar.khaled@gmail.com',
    level: 'Junior',
    role: 'student',
    postsCount: 1,
    commentsCount: 2,
    reactionsGivenCount: 2,
    reactionsReceivedCount: 1,
    questionsAsked: 1,
    announcementsMade: 0,
    contributionScore: 13,
    popularityScore: 1,
    memberSince: 2024-02-01T00:00:00.000Z
  }
```

Note: This screenshot does not contain all the documents, just samples.

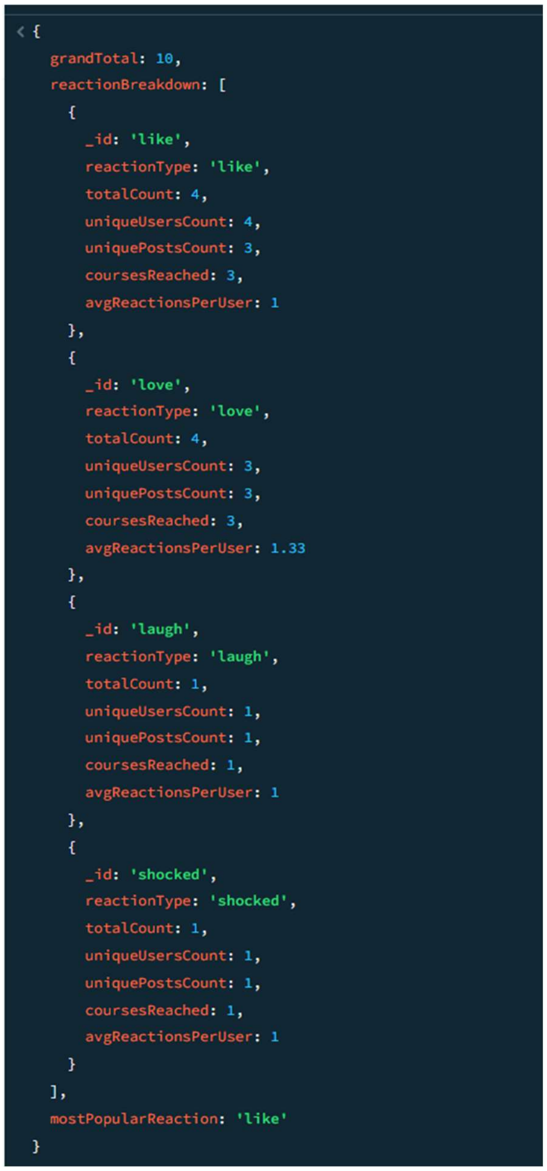
2.5.3 pipeline 3: reaction distribution analysis

Collection: reactions.

Purpose: Analyse reaction types distribution across posts and time.

```
db.reactions.aggregate([
  {
    $group: {
      _id: "$type",
      count: { $sum: 1 },
      uniqueUsers: { $addToSet: "$senderId" },
      uniquePosts: { $addToSet:  "$postId" }
    }
  },
  {
    $lookup: {
      from: "posts",
      localField: "uniquePosts",
      foreignField: "_id",
      as: "postDetails"
    }
  },
  {
    $project: {
      reactionType: "$_id",
      totalCount: "$count",
      uniqueUsersCount: { $size: "$uniqueUsers" },
      uniquePostsCount: { $size: "$uniquePosts" },
      coursesReached: {
        $size: {
          $setUnion: {
            $map: {
              input: "$postDetails",
              as: "post",
              in: "$$post.courseId"
            }
          }
        }
      },
      avgReactionsPerUser: {
        $round: [
          { $divide: ["$count", { $size: "$uniqueUsers" }] },
          2
        ]
      }
    }
  },
  {$sort: { totalCount: -1 }},
  {$group: {
      _id: null,
      reactions: { $push: "$$ROOT" },
      grandTotal: { $sum: "$totalCount" }
    }
  },
  {$project: {
      _id: 0,
      grandTotal: 1,
      reactionBreakdown: "$reactions",
      mostPopularReaction: { $arrayElemAt: ["$reactions.reactionType", 0] }
    }
  }
])
```

Output:



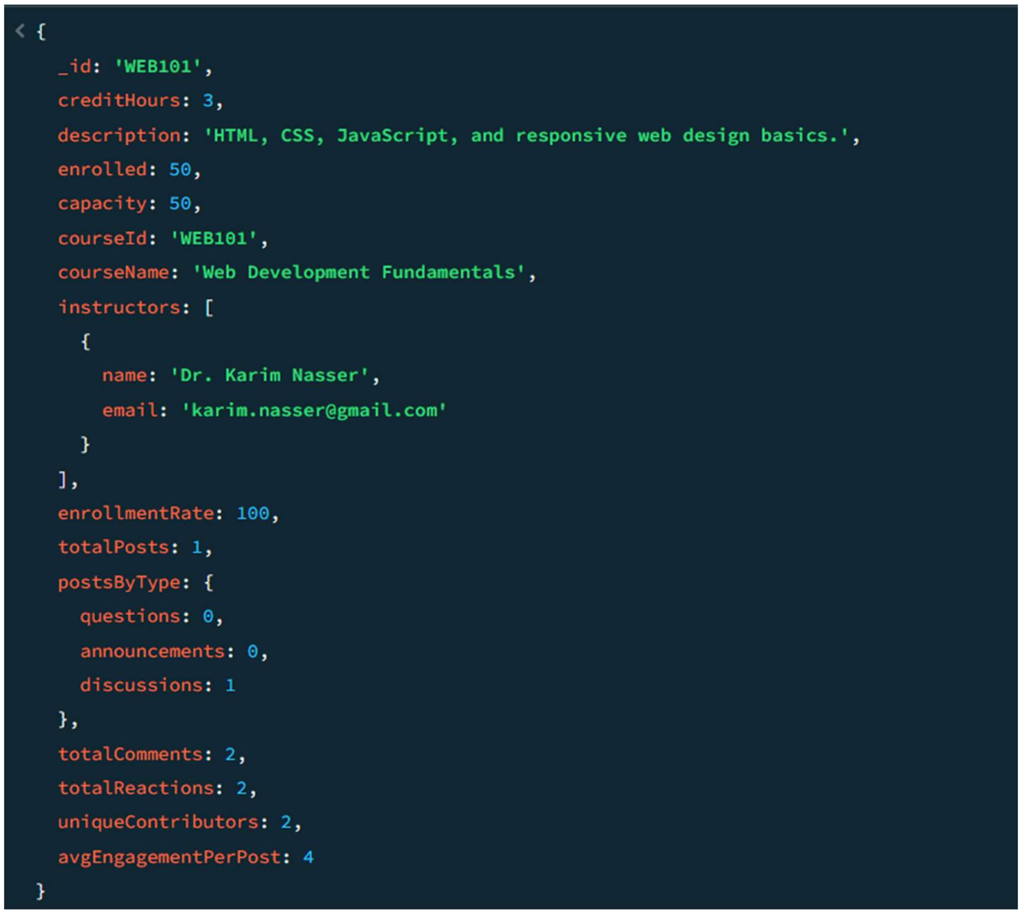
```
< {
    grandTotal: 10,
    reactionBreakdown: [
      {
        _id: 'like',
        reactionType: 'like',
        totalCount: 4,
        uniqueUsersCount: 4,
        uniquePostsCount: 3,
        coursesReached: 3,
        avgReactionsPerUser: 1
      },
      {
        _id: 'love',
        reactionType: 'love',
        totalCount: 4,
        uniqueUsersCount: 3,
        uniquePostsCount: 3,
        coursesReached: 3,
        avgReactionsPerUser: 1.33
      },
      {
        _id: 'laugh',
        reactionType: 'laugh',
        totalCount: 1,
        uniqueUsersCount: 1,
        uniquePostsCount: 1,
        coursesReached: 1,
        avgReactionsPerUser: 1
      },
      {
        _id: 'shocked',
        reactionType: 'shocked',
        totalCount: 1,
        uniqueUsersCount: 1,
        uniquePostsCount: 1,
        coursesReached: 1,
        avgReactionsPerUser: 1
      }
    ],
    mostPopularReaction: 'like'
  }
```

2.5.4 pipeline 4: instructor course performance report

Collection: courses.
Purpose: Detailed analytics for instructor's courses with student engagement.
Note: We’ll replace ObjectId("INSTRUCTOR_ID_HERE") with actual instructor ID.

```
db.courses.aggregate([
  // Match courses by instructor (replace with actual ObjectId)
  // {
  //   $match: {
  //     instructorId: ObjectId("INSTRUCTOR_ID_HERE")
  //   }
  // },
  // Lookup all posts in each course
  {
    $lookup: {
      from: "posts",
      localField: "_id",
      foreignField: "courseId",
      as: "coursePosts"
    }
  },
  {
    $lookup: {
      from: "comments",
      localField: "coursePosts._id",
      foreignField: "postId",
      as: "courseComments"
    }
  },
  {
    $lookup: {
      from: "reactions",
      localField: "coursePosts._id",
      foreignField: "postId",
      as: "courseReactions"
    }
  },
  {
    $lookup: {
      from: "users",
      localField: "instructorId",
      foreignField: "_id",
      as: "instructorInfo"
    }
  },
  {
    $project: {
      courseId: "$_id",
      courseName: "$name",
      description: 1,
      creditHours: 1,
      enrolled: 1,
      capacity: 1,
      instructors: {
        $map: {
          input: "$instructorInfo",
          as: "inst",
          in: { name: "$$inst.name", email: "$$inst.email" }
        }
      },
      enrollmentRate: {
        $round: [
          {
            $multiply: [
              { $divide: ["$enrolled", { $max: ["$capacity", 1] }] },
              100
            ]
          },
          1
        ]
      },
      totalPosts: { $size: "$coursePosts" },
      postsByType: {
        questions: {
          $size: {
            $filter: {
              input: "$coursePosts",
              as: "p",
              cond: { $eq: ["$$p.type", "question"] }
            }
          }
        },
        announcements: {
          $size: {
            $filter: {
              input: "$coursePosts",
              as: "p",
              cond: { $eq: ["$$p.type", "announcement"] }
            }
          }
        },
        discussions: {
          $size: {
            $filter: {
              input: "$coursePosts",
              as: "p",
              cond: { $eq: ["$$p.type", "discussion"] }
            }
          }
        }
      },
      totalComments: { $size: "$courseComments" },
      totalReactions: { $size: "$courseReactions" },
      uniqueContributors: {
        $size: {
          $setUnion: [
            { $map: { input: "$coursePosts", as: "p", in: "$$p.sender.id" } },
            { $map: { input: "$courseComments", as: "c", in: "$$c.sender.id" }
}
          ]
        }
      },
      avgEngagementPerPost: {
        $cond: [
          { $eq: [{ $size: "$coursePosts" }, 0] },
          0,
          {
            $round: [
              {
                $divide: [
                  { $add: [{ $size: "$courseComments" }, { $size:
"$courseReactions" }] },
                  { $size: "$coursePosts" }
                ]
              },
              2
            ]
          }
        ]
      }
    }
  },
  {
    $sort: { enrolled: -1 }
  }
])
```

Output:

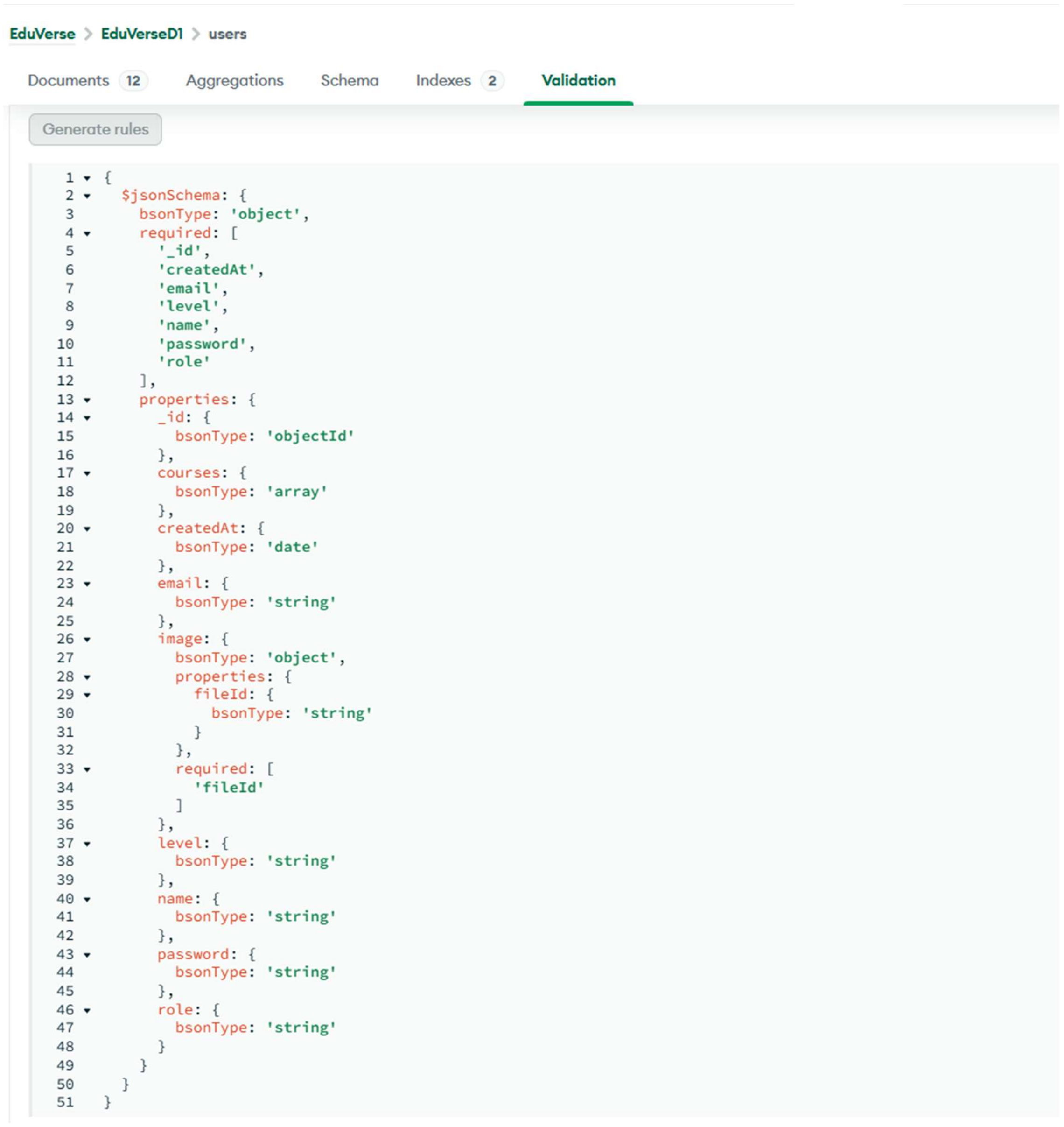


```
< {
    _id: 'WEB101',
    creditHours: 3,
    description: 'HTML, CSS, JavaScript, and responsive web design basics.',
    enrolled: 50,
    capacity: 50,
    courseId: 'WEB101',
    courseName: 'Web Development Fundamentals',
    instructors: [
      {
        name: 'Dr. Karim Nasser',
        email: 'karim.nasser@gmail.com'
      }
    ],
    enrollmentRate: 100,
    totalPosts: 1,
    postsByType: {
      questions: 0,
      announcements: 0,
      discussions: 1
    },
    totalComments: 2,
    totalReactions: 2,
    uniqueContributors: 2,
    avgEngagementPerPost: 4
  }
```

Note: This screenshot does not contain all the documents, just samples.

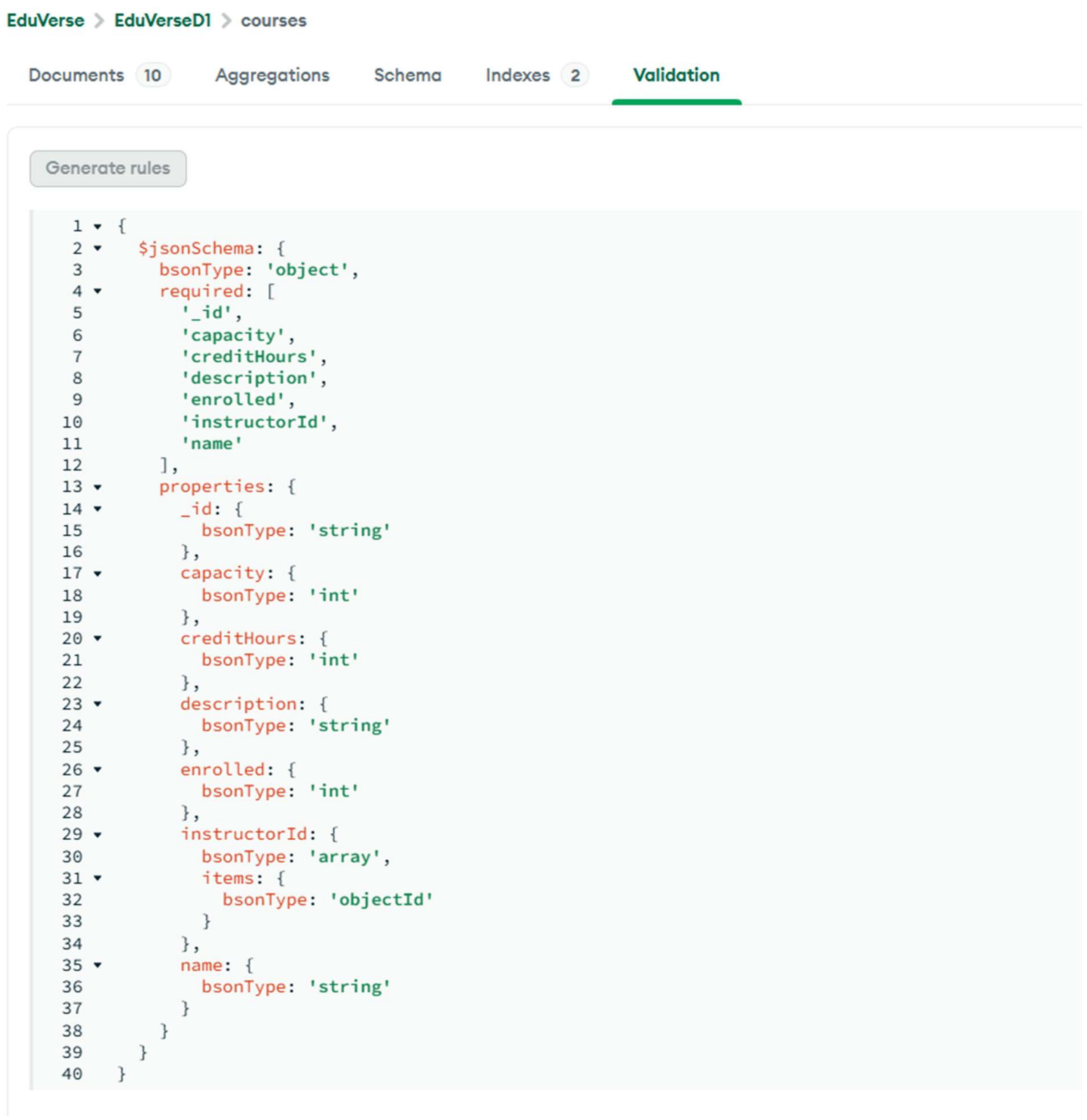
2.6 Schema Validation

2.6.1 json schema validation for the users collection



```
{
  $jsonSchema: {
    bsonType: 'object',
    required: [
      '_id',
      'createdAt',
      'email',
      'level',
      'name',
      'password',
      'role'
    ],
    properties: {
      _id: {
        bsonType: 'objectId'
      },
      courses: {
        bsonType: 'array'
      },
      createdAt: {
        bsonType: 'date'
      },
      email: {
        bsonType: 'string'
      },
      image: {
        bsonType: 'object',
        properties: {
          fileId: {
            bsonType: 'string'
          }
        },
        required: [
          'fileId'
        ]
      },
      level: {
        bsonType: 'string'
      },
      name: {
        bsonType: 'string'
      },
      password: {
        bsonType: 'string'
      },
      role: {
        bsonType: 'string'
      }
    }
  }
}
```

2.6.2 json schema validation for the courses collection



```
{
  $jsonSchema: {
    bsonType: 'object',
    required: [
      '_id',
      'capacity',
      'creditHours',
      'description',
      'enrolled',
      'instructorId',
      'name'
    ],
    properties: {
      _id: {
        bsonType: 'string'
      },
      capacity: {
        bsonType: 'int'
      },
      creditHours: {
        bsonType: 'int'
      },
      description: {
        bsonType: 'string'
      },
      enrolled: {
        bsonType: 'int'
      },
      instructorId: {
        bsonType: 'array',
        items: {
          bsonType: 'objectId'
        }
      },
      name: {
        bsonType: 'string'
      }
    }
  }
}
```

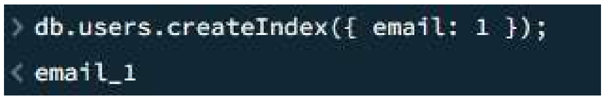
NOTE: Rest of Validations can be shown inside the database.

2.7 Indexing Strategy

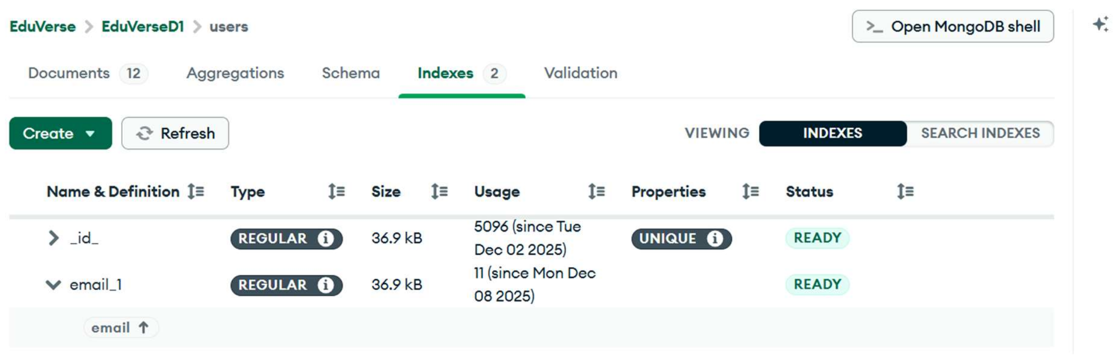
Collection	Index	Type	Purpose
Users	email	Unique	Fast login lookups, prevent duplicates
Posts	courseId	Regular	Filter posts by course
Posts	sender.id	Regular	Get user's posts
Posts	createdAt	Regular	Sort by date
Comments	postId	Regular	Get comments for a post
Reactions	postId, senderId	Compound	Check user's reaction on post
Messages	senderId, receiverId	Compound	Get conversation messages
Chats	user1.id, user2.id	Compound	Find existing chat

2.7.1 creating an index at the users collection

Command: db.users.createIndex({ email: 1 });



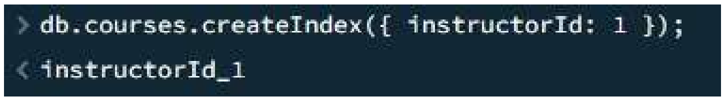
```
> db.users.createIndex({ email: 1 });
< email_1
```



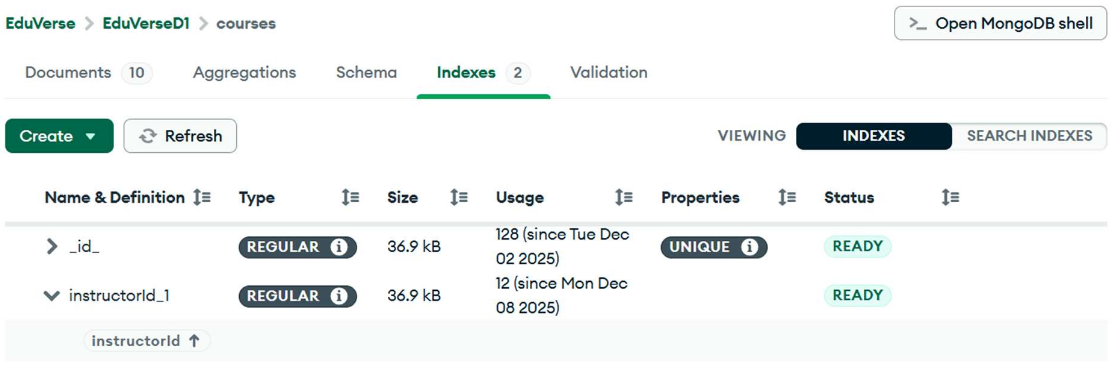
Name & Definition	Type	Size	Usage	Properties	Status
_id_	REGULAR	36.9 kB	5096 (since Tue Dec 02 2025)	UNIQUE	READY
email_1	REGULAR	36.9 kB	11 (since Mon Dec 08 2025)		READY
email ↑					

2.7.2 creating an index at the courses collection

Command: db.courses.createIndex({ instructorId: 1 });



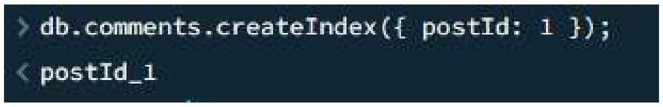
```
> db.courses.createIndex({ instructorId: 1 });
< instructorId_1
```



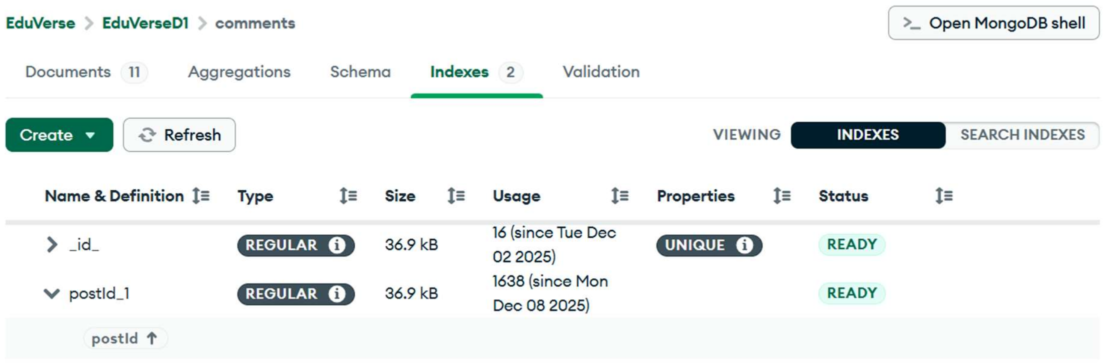
Name & Definition	Type	Size	Usage	Properties	Status
_id_	REGULAR	36.9 kB	128 (since Tue Dec 02 2025)	UNIQUE	READY
instructorId_1	REGULAR	36.9 kB	12 (since Mon Dec 08 2025)		READY
instructorId ↑					

2.7.3 creating an index at the comments collection

Command: db.comments.createIndex({ postId: 1 });



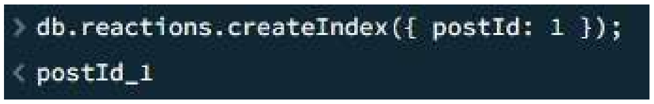
```
> db.comments.createIndex({ postId: 1 });
< postId_1
```



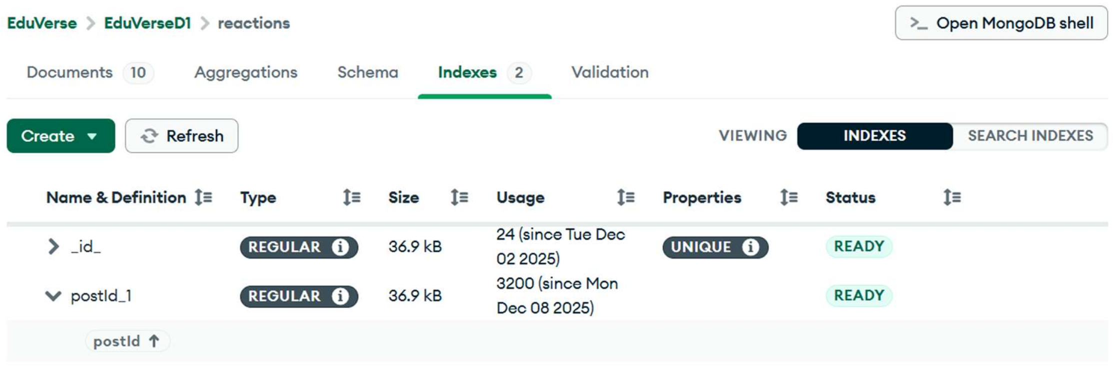
Name & Definition	Type	Size	Usage	Properties	Status
_id_	REGULAR	36.9 kB	16 (since Tue Dec 02 2025)	UNIQUE	READY
postId_1	REGULAR	36.9 kB	1638 (since Mon Dec 08 2025)		READY
postId ↑					

2.7.4 creating an index at the reactions collection

Command: db.reactions.createIndex({ postId: 1 });



```
> db.reactions.createIndex({ postId: 1 });
< postId_1
```



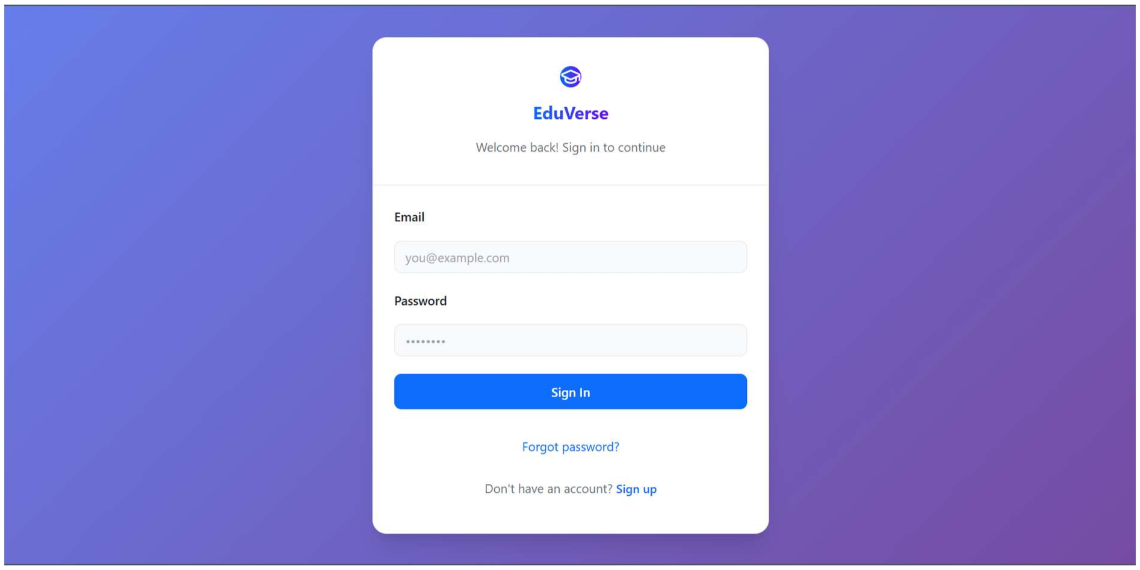
Name & Definition	Type	Size	Usage	Properties	Status
_id_	REGULAR	36.9 kB	24 (since Tue Dec 02 2025)	UNIQUE	READY
postId_1	REGULAR	36.9 kB	3200 (since Mon Dec 08 2025)		READY

postId ↑

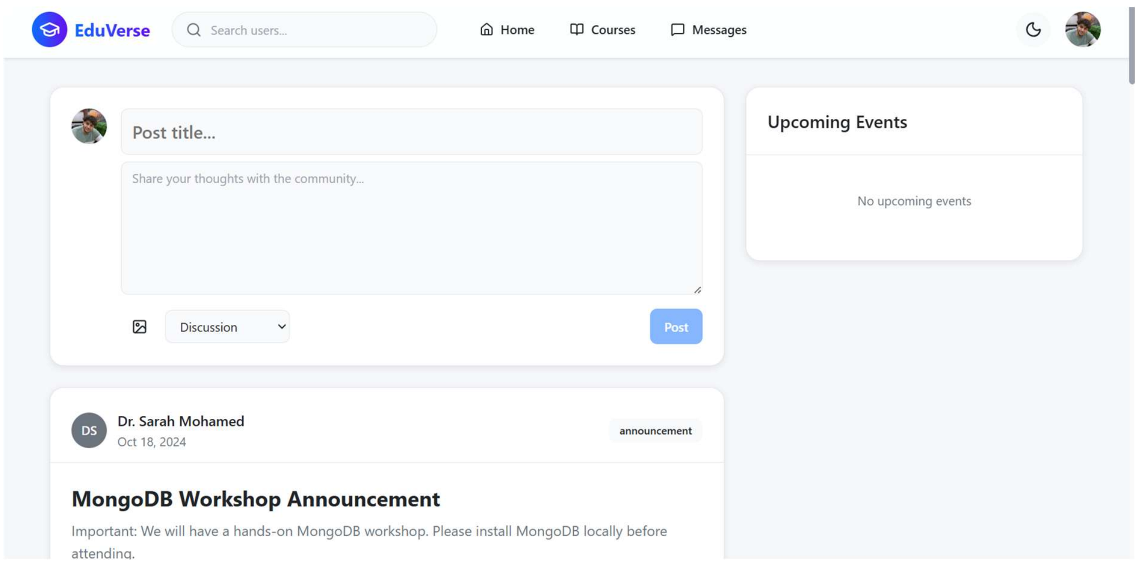
NOTE: **Rest of Indexes can be shown inside the database.**

3.0 WEBSITE FUNCTIONALITY

3.1 Signup/Login Page

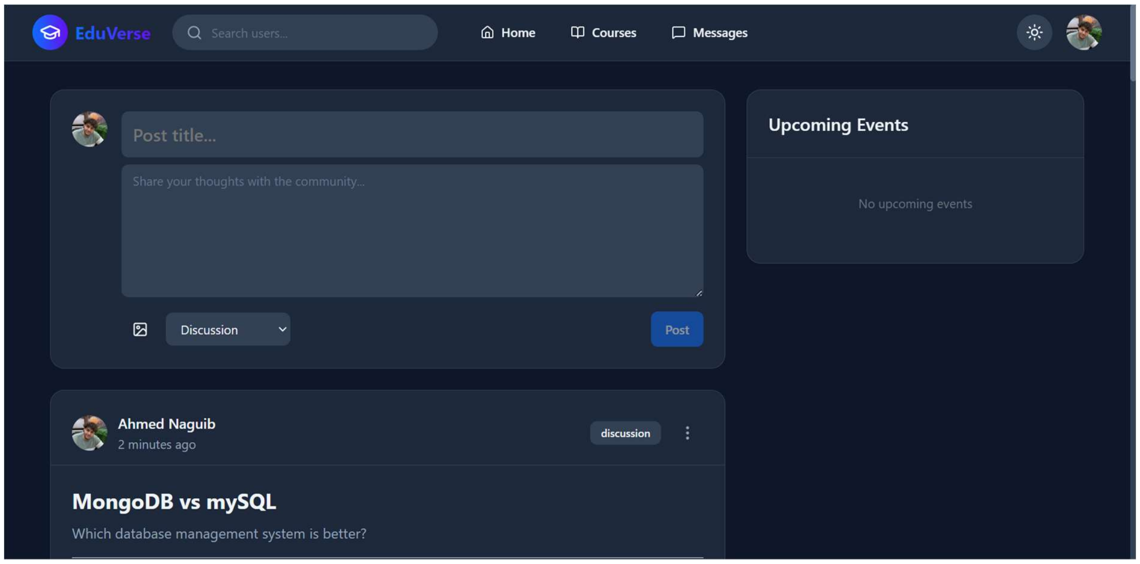


3.2 Home Page



MongoDB Workshop Announcement

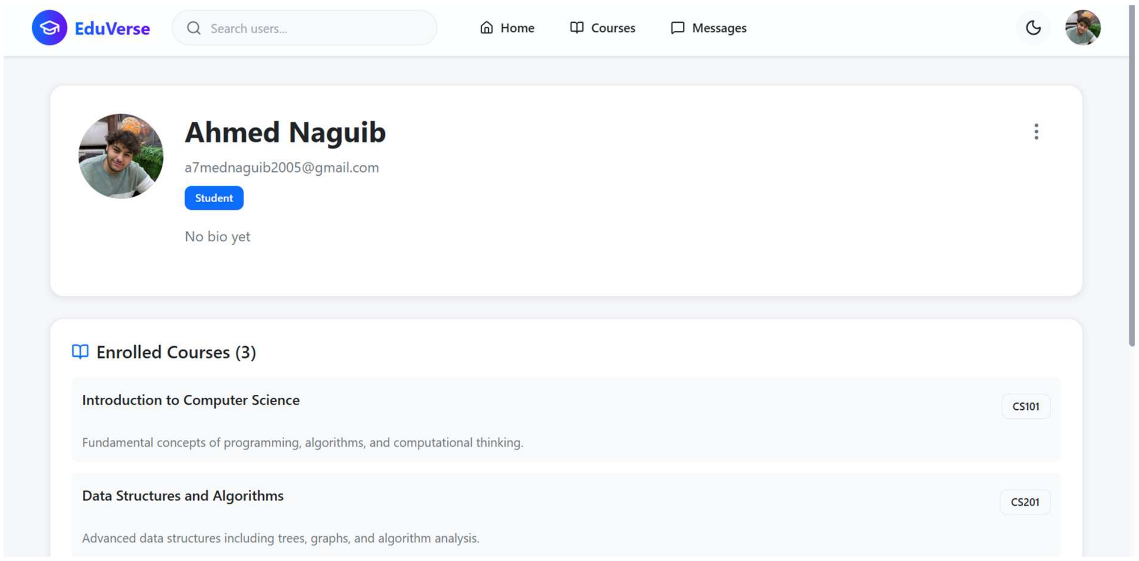
Important: We will have a hands-on MongoDB workshop. Please install MongoDB locally before attending.



Ahmed Naguib
2 minutes ago
discussion

MongoDB vs mySQL
Which database management system is better?

3.3 Profile Page

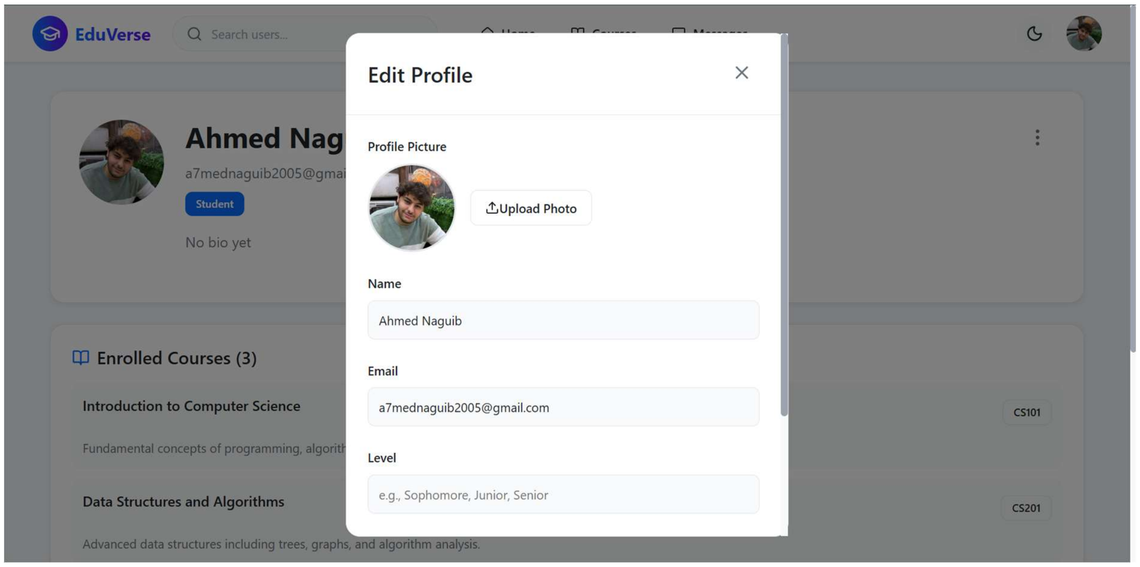


Ahmed Naguib
a7mednaguib2005@gmail.com
Student
No bio yet

Enrolled Courses (3)

Introduction to Computer Science CS101
Fundamental concepts of programming, algorithms, and computational thinking.

Data Structures and Algorithms CS201
Advanced data structures including trees, graphs, and algorithm analysis.



EduVerse

Edit Profile

Profile Picture

Upload Photo

Name

Ahmed Naguib

Email

a7mednaguib2005@gmail.com

Level

e.g., Sophomore, Junior, Senior

Ahmed Nag

a7mednaguib2005@gmai

Student

No bio yet

Enrolled Courses (3)

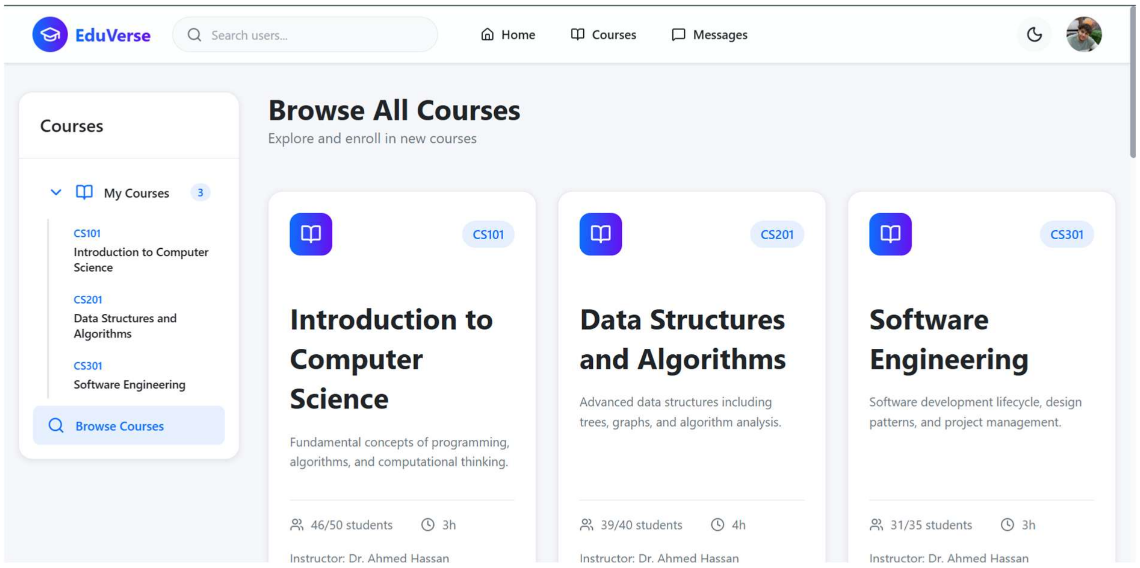
Introduction to Computer Science CS101

Fundamental concepts of programming, algorith

Data Structures and Algorithms CS201

Advanced data structures including trees, graphs, and algorithm analysis.

3.4 Courses Page



EduVerse | Search users... | Home | Courses | Messages

Courses

My Courses 3

CS101 Introduction to Computer Science

CS201 Data Structures and Algorithms

CS301 Software Engineering

Browse Courses

Browse All Courses

Explore and enroll in new courses

CS101 Introduction to Computer Science

Fundamental concepts of programming, algorithms, and computational thinking.

46/50 students 3h

Instructor: Dr. Ahmed Hassan

CS201 Data Structures and Algorithms

Advanced data structures including trees, graphs, and algorithm analysis.

39/40 students 4h

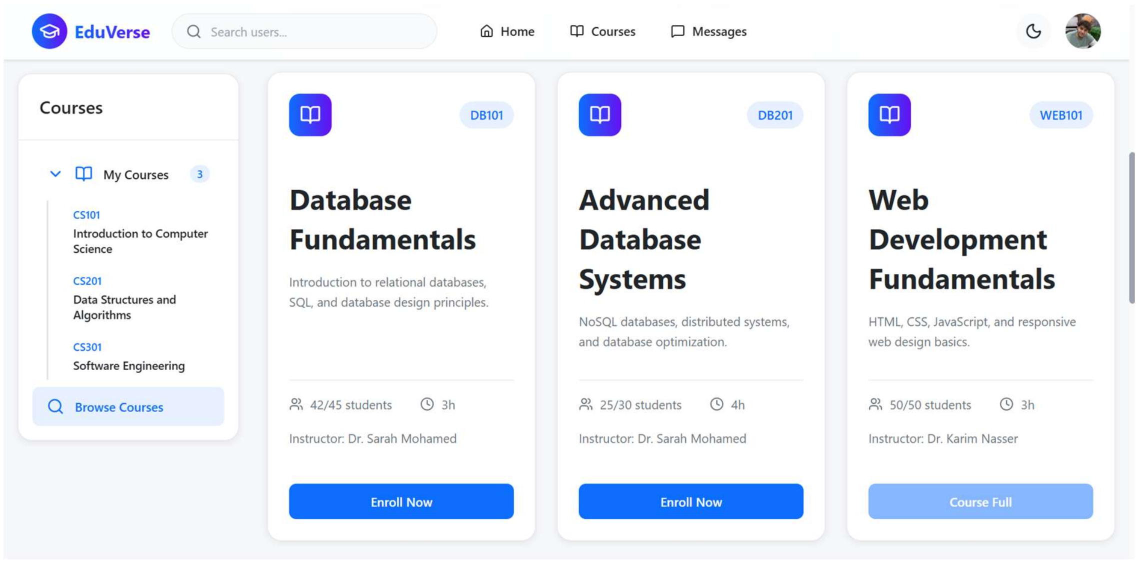
Instructor: Dr. Ahmed Hassan

CS301 Software Engineering

Software development lifecycle, design patterns, and project management.

31/35 students 3h

Instructor: Dr. Ahmed Hassan



Courses

My Courses 3

CS101
Introduction to Computer Science

CS201
Data Structures and Algorithms

CS301
Software Engineering

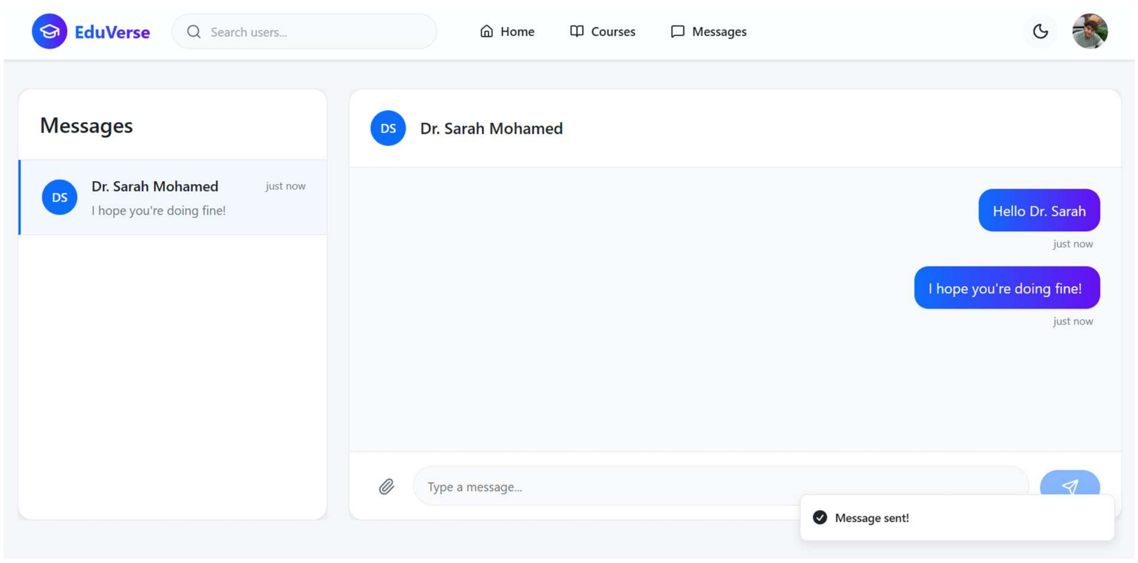
Browse Courses

DB101
Database Fundamentals
Introduction to relational databases, SQL, and database design principles.
42/45 students 3h
Instructor: Dr. Sarah Mohamed
Enroll Now

DB201
Advanced Database Systems
NoSQL databases, distributed systems, and database optimization.
25/30 students 4h
Instructor: Dr. Sarah Mohamed
Enroll Now

WEB101
Web Development Fundamentals
HTML, CSS, JavaScript, and responsive web design basics.
50/50 students 3h
Instructor: Dr. Karim Nasser
Course Full

3.5 Chat Page



Messages

DS Dr. Sarah Mohamed just now
I hope you're doing fine!

DS Dr. Sarah Mohamed

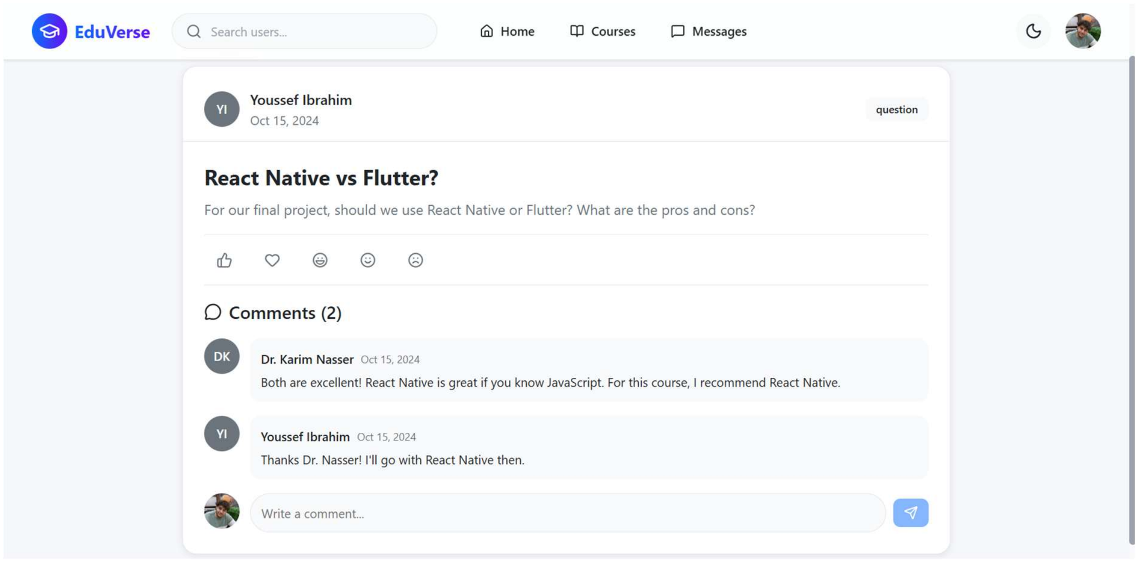
Hello Dr. Sarah
just now

I hope you're doing fine!
just now

Type a message...

Message sent!

3.6 Viewing a Post



EduVerse | Search users... | Home | Courses | Messages

Youssef Ibrahim
Oct 15, 2024
question

React Native vs Flutter?

For our final project, should we use React Native or Flutter? What are the pros and cons?

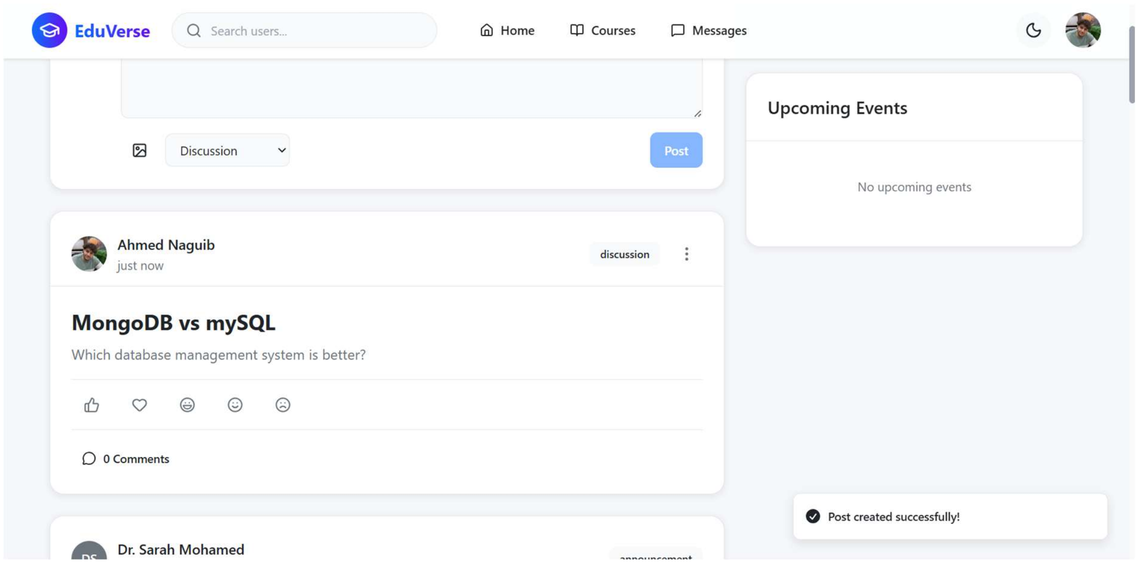
Comments (2)

Dr. Karim Nasser Oct 15, 2024
Both are excellent! React Native is great if you know JavaScript. For this course, I recommend React Native.

Youssef Ibrahim Oct 15, 2024
Thanks Dr. Nasser! I'll go with React Native then.

Write a comment...

3.7 Creating a Post



EduVerse | Search users... | Home | Courses | Messages

Discussion | Post

Ahmed Naguib
just now
discussion

MongoDB vs mySQL

Which database management system is better?

0 Comments

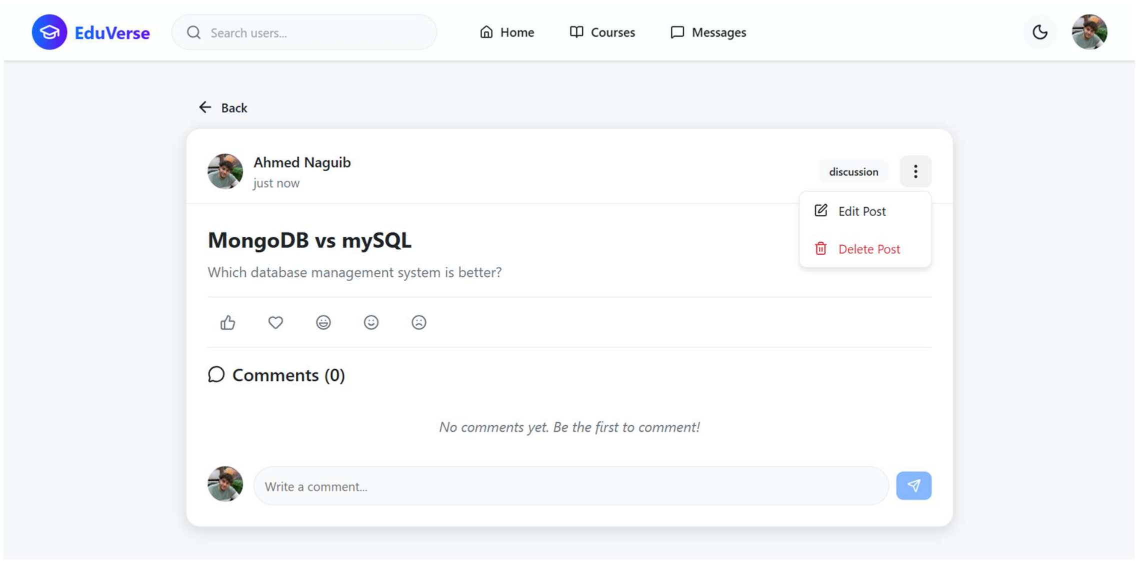
Dr. Sarah Mohamed

Upcoming Events

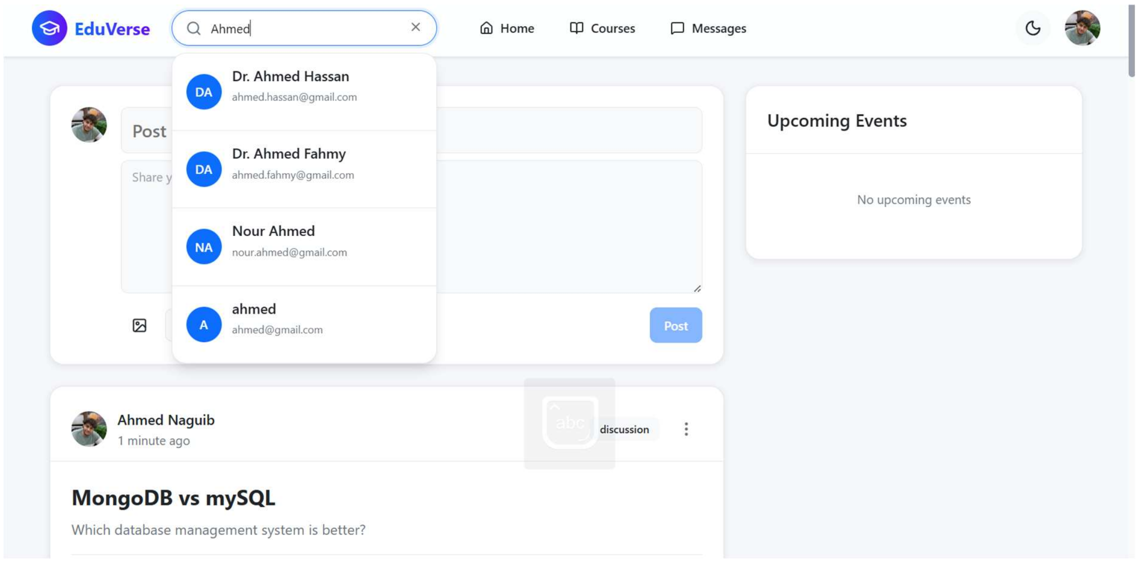
No upcoming events

Post created successfully!

3.8 Editing/Deleting a Post



3.9 Searching


EduVerse
Ahmed
Home
Courses
Messages
Dr. Ahmed Hassan
ahmed.hassan@gmail.com
Dr. Ahmed Fahmy
ahmed.fahmy@gmail.com
Nour Ahmed
nour.ahmed@gmail.com
ahmed
ahmed@gmail.com
Post
Upcoming Events
No upcoming events
Ahmed Naguib
1 minute ago
discussion
MongoDB vs mySQL
Which database management system is better?