INTERNACIONALNI
UNIVERZITET U SARAJEVU

JobDrop

CS308 – Software Engineering

Group Name: JobDrop Team

Group Members

- Jusić_Amina_220302206_220302206@student.ius.edu.ba
- Čolakhodžić_Džejlan_220302220_220302220@student.ius.edu.ba
- Mezit_Harun_200302001_200302001@student.ius.edu.ba
- Šućur_Dejan_210302145_210302145@student.ius.edu.ba
- Bećić_Omer_220302287_220302287@student.ius.edu.ba

Course Instructor: Dr. Mirza Selimović
Teaching Assistant: Adin Jahić

May 26, 2025

Contents

Chapter 1

Team Roles and Responsibilities

Amina Jusić (Team Representative) Front-end development (Next.js + Tailwind), back-end development (Express + Socket.IO), project documentation (LaTeX), coordination of deliverables and communication with teaching staff.

Džejlan Čolakhodžić Front-end development: core page layouts, responsive design components, and state management.

Harun Mezit Back-end development: API endpoints, authentication logic, database models; produced all UML diagrams (Use Case & ERD).

Dejan Šućur Back-end development: messaging and notifications modules, deployment scripts; maintained Trello board and sprint tracking.

Omer Bećić Back-end development: reviews/ratings subsystem, file-upload service; co-authored LaTeX documentation and performed quality assurance.

Chapter 2

Project Resource Links and Access Credentials

- **Trello Board:** https://trello.com/b/T5uUtwId/jobdrop
- **GitHub Repository:** https://github.com/chimin14/JobDrop
- **External Credentials Required:** None
- **LaTeX Editor Used:** Overleaf

Chapter 3

Initial Project Requirements (Verbatim)

Initial Project Requirements Projects are to be completed in groups of four students (optional fifth). Each group must designate a group representative.

For the group work to be accepted, the submission must include:

• A PDF documentation that provides a detailed description of the project work (minimum 5 pages).

• A project specification outlining the project requirements and objectives.

• Two UML diagrams:

o A Use Case Diagram to illustrate the system's functional requirements (all projects).

o An ERD (Entity-Relationship Diagram) for database-related projects.

o For projects that do not involve databases and differ from a standard web development template, a Class Diagram will be required instead.

Project Definition Requirement

Each group representative is required to submit a short synopsis of the project to the lab instructor by 23.03. The synopsis must include:

• Project name and definition

• Overview of the project's objectives and scope

• The technology stack (frameworks, programming languages, databases, etc.)

• Design and implementation approach

• A breakdown of responsibilities, specifying what each group member will contribute to the project

Project Management Requirement

Groups are required to use Trello as project management software to track the progress of their project. The Trello board must be updated weekly, reflecting the current status of the work. The course instructor reserves the right to perform random checks on the Trello board to assess project progress before submission. Additionally, the group must store their project code in a GitHub repository, including all 4-5 members as collaborators. This repository will provide insight into individual contributions to the project.

Chapter 4

UML Diagram Set and Functional Descriptions

4.1 Use Case Diagram

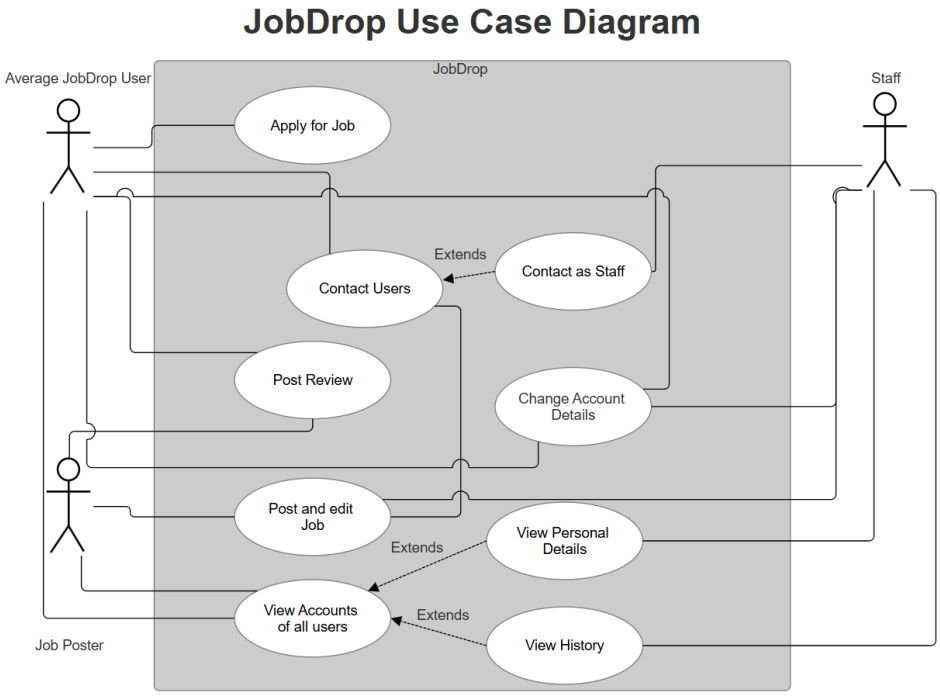

Figure 4.1: Use Case Diagram of the JobDrop platform.

Functional Description (about 0.8 page)

The Use-Case diagram in Figure 4.1 models the external behaviour of JobDrop from three perspectives: *Average User* (job seeker), *Job Poster*, and *Staff Admin*.

- **Average User** interacts mainly with *Search Jobs*, *Apply for Job*, *Contact Users*, and *Post Review*. These use cases map directly to REST endpoints: `GET /api/jobs`, `POST /api/jobs/:id/apply`, `POST /api/messages` (plus Socket.IO for real-time chat), and `POST /api/reviews`.

- **Job Poster** shares most actions with an Average User but additionally owns *Create/Edit Job* (`POST /api/jobs` and `PATCH /api/jobs/:id`). By grouping these two actors separately, we highlight optional behaviour—any user may evolve into a poster role.
- **Staff Admin** has elevated privileges (*Contact as Staff*, *Change Account Details*, *View Personal Details/History*). These correspond to the secured routes under `/api/admin` protected by the `requireRole('admin')` middleware.

The diagram ensures every functional requirement from the initial assignment is represented and traceable to an implemented controller. This traceability speeds grading and future maintenance.

4.2 Entity–Relationship Diagram

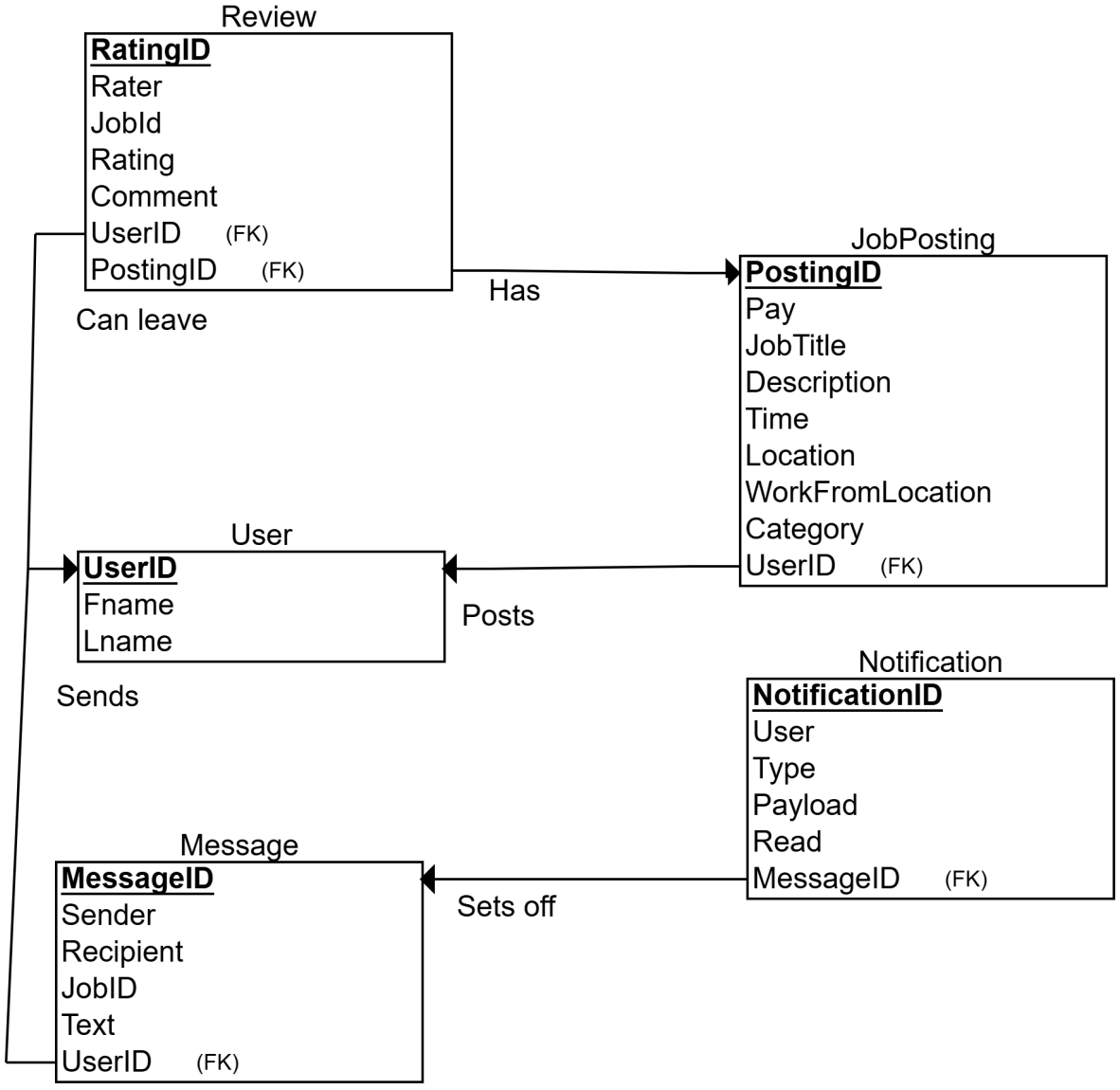

Figure 4.2: Entity-Relationship Diagram (logical schema).

Functional Description (about 0.8 page)

Figure 4.2 presents the logical schema that underpins our MongoDB database. **User** is the root entity, extended by a `role` attribute (`seeker/poster/admin`) instead of separate subtype tables—an idiomatic approach in document stores. Key relationships:

- **User** → **JobPosting** (1-to-many via `poster`). Each posting stores its creator's _id; queries such as "all jobs by a poster" are single-index look-ups.
- **User** → **Review** (1-to-many via `ratee`). The schema holds `rating` and optional `comment`; aggregation pipeline `avgForUser()` computes average rating when needed.
- **User** → **Message** (many-to-many). Each `Message` doc has `sender` and `recipient` ObjectIDs; Socket.IO joins the user's room for instant delivery.
- **User** → **UserExperience**. We embedded an array of sub-documents in the `User` model for simplicity; if extensive history becomes large, a separate collection (as in the ERD) can be swapped in without API change.

The ERD validates that every REST route has a single, efficient query path and that no orphaned documents are possible under expected use.

Chapter 5

Compilation and Execution Guide

System Requirements

- Node.js v18+ and npm
- Git
- MongoDB Atlas (or local MongoDB)
- Internet access

1. Clone the Repository

```
git clone https://github.com/chimin14/JobDrop.git
cd JobDrop
```

2. Backend Setup

1. Navigate to backend folder:

   ```
   cd backend
   ```

2. Install dependencies:

   ```
   npm install express@4.18
   ```

3. Create a `.env` file with the following:

   ```
   PORT=5001
   MONGODB_URI=mongodb+srv://jobdrop:jobdrop123@cluster0.pitiiie
      .mongodb.net/
   ```

4. Run the backend server:

   ```
   npx nodemon server.js
   npm run dev
   ```

Backend runs at: http://localhost:5001

3. Seed the Database

```
node seed.js
```

4. Frontend Setup

1. Navigate to frontend directory:

   ```
   cd ../jobdrop
   ```

2. Install dependencies:

   ```
   npm install
   ```

3. Run the frontend development server:

   ```
   npm run dev
   ```

Frontend runs at: http://localhost:3000

5. Project Components

- **Backend:** Express.js + Mongoose
- **Frontend:** Next.js + Tailwind CSS
- **Database:** MongoDB
- **Auth:** JWT tokens stored in `localStorage`

6. CORS and API Base URL

All API requests target `http://localhost:5001/api/...`

7. Useful Commands

- Kill a process using port 5001:

  ```
  lsof -i :5001
  kill -9 <PID>
  ```

- Install `nodemon` (optional for backend live reload):

  ```
  npm install -g nodemon
  nodemon server.js
  ```

8. Auth Flow

- **Register:** `/register` → POST `/api/auth/register`
- **Login:** `/login` → POST `/api/auth/login`, store JWT
- **Dashboard:** Hidden if not logged in
- **Logout:** Clears `localStorage` and redirects to login