Freelance Job Board

This project is about creating a web application for companies to post projects and people who wish to work on them can bid and apply to the wanted job. Payment tracking system will also be implemented into this web application.

Required core entities

- **Users** – individuals registered in the system, each assigned to specific roles (many for a user).
- Employer – job listings, freelancer profiles, job posting, payment integration
- Job – Iduser, name, email, role, …

User Roles

1. **Administrator** – responsible for managing user accounts, roles, and overall system configuration.
2. **Management** – authorized to create changes, assign types to users, and oversee regular users activity. //select user for available jobs
3. Regular User
    - Employer – posts available jobs and opens the ability to bid for a work position
    - Freelancer – configures user settings and can apply for a job
    - Reviewer – trusted person who reviews completed jobs or validates the freelancer' performances
    - Support – user who is responsible for assisting other users through support tickets or FAQ
4. **Unregistered or Control User** - can look at job offerings but without possibility of change.

Key Features

- Role-based permissions to ensure secure and structured access control in both domains: frontend human interaction and rest endpoints usage.
- Creation of categories and assignment of managers.
- Centralized **SQLite** database to store and manage all users and other entities
- User authentication for secure login and registration
- Automatic logging of critical actions like job creation, role changes, if the job was taken…
- Assignment tracking for a user to see the status changes, timestamps, …
- Job lifecycle management that includes status updates, deadlines and progress visibility
- Sorting system that will make searching for a job easier and more organized

Business Rules

The system enforces business rules based on user roles. Each role grants specific responsibilities and restrictions to ensure normal flow in the context of your project.

Administrator

- Manages **users** within the system (e.g. login/emails, passwords etc.).
- Assigns and updates **roles** for each user.
- A user can hold **one role (administrator, management, regular user or unregistered/control user)**

Management

- Creates and modifies job settings
- Assigns types of users **in the project context** from the pool of Regular Users. A regular user can have multiple types at the same time.
- Can reassign/change the type if necessary.
- Does not directly create or manage lower functionality of the project.

Regular User

Depending on their type in a given, a Regular User may act as:

1. Employer:
    - Post and manage job listings
    - Review freelancer proposals
    - Assign jobs to freelancers
    - Approve submissions and payments
    - Leave reviews after project completion
2. Freelancer:
    - Create and edit personal profile
    - Browse and apply to job postings
    - Submit proposals, manage assigned tasks.
    - Track deadlines and update status
    - Communicate with clients
3. Reviewer:
    - Access job completion reports
    - Provide quality feedback or approval

        - Cannot assign or edit user roles

4. Support user:
    - View and respond to support requests
    - Escalate issues to management or admin
    - No access to user or system configuration

Unregistered or Control User

- Unregistered user can view and browse job offerings and look at the detailed view of them
- Creating an account

Common features required by all projects

Data Integrity and Validation

- All mandatory fields (e.g., user name, project title, task description) must be validated before saving.
- Unique identifiers (e.g., usernames, project IDs) must be enforced at the database level.
- Relationships between entities must be preserved.
- Invalid or incomplete data cannot be persisted.

Usability and User Experience

- The system must provide a clear, responsive web interface accessible across modern browsers.
- Actions should be simple and intuitive, following established UI/UX conventions.
- Error messages must be descriptive, guiding the user on how to resolve issues.
- **Large datasets -** are displayed in a way that is user-friendly and does not overuse browser resources (e.g. pagination on server side)
- **Sorting, filtering** - Filtering and sorting must be done on the server side.
- **Technologies** - must be implemented using **Angular**, data is retrieved from the backed **REST service** according to the standards. Using Angular material is allowed.
- Data should be presented graphically (charts) in at least two places.

Security and Access Control

- Authentication is required for all users, except for the unregistered user who has limited access to only view some small portions of the things in the context of the project.
- Authorization is role-based, ensuring users only access functions permitted by their role(s) on backend API. The frontend is tailored to the backend's capabilities within a given set of roles.
- Multiple types per registered user are supported, with the system applying the union of their permissions.
- Sensitive actions (e.g., user role changes, project manager reassignment) require confirmations to ensure user intentions.
- Passwords must be stored in a secure form not plain text.

Audit and Traceability

- All critical changes (user role updates, project manager reassignments, task status approvals) must be logged.
- The system should provide visibility into who performed specific actions and when.
- A special GUI for log browsing is not obligatory.

Other requirements

- As a starting point teachers' demo projects can be used.
- **Running the application on a production server -** NodeJS server started on a port number given by teachers to the student. Backed must have a necessary **REST** service that provides access to the resources needed by the frontend. Backend also hosts the angular application. The application is loaded when the user opens the root of the project in the browser (e.g. http://spider.foi.hr:12000) Every communication is then handled via **recommended standards**. REST service implementation should follow the conventional implementation using HTTP methods for specific purposes. (i.e. GET - retrieve data, POST - add data, DELETE - deleting data, and PUT - update of data)
    - **Installation and modules** - project must be installed on the server spider.foi.hr. Access via the SSH must be restricted to all other students.
    - **Frameworks/modules/libraries**: In the assignment, only the frameworks/modules/libraries that have been covered in class are allowed to be used on both the client-side and server-side. Specifically for NodeJS, only those modules that are globally installed on the server spider.foi.hr may be used, and no additional modules may be installed on the server-side. A list of available modules can be obtained with: `npm -g list`.
    - **Assignment structure**: The directory should only contain files and subdirectories related to the assignment; everything else must be removed (e.g., scripts from exercises). Relative paths must be used throughout the assignment. Multimedia files uploaded to the server must adhere to the maximum size limits: 500KB for images and 1MB for videos!
- **Help and Questions:** All questions regarding the assignment should be posted in the dedicated assignment forum on the Moodle system. You can also ask the instructor during lab sessions or during consultations scheduled by the instructor at the respective institution. We recommend that you do not share your code with other teams, do not show your code to other teams, and do not work on the assignment together with other teams, even if you are roommates or have known each other for 10 years. Copying assignments is prohibited; if it is determined that the assignment was copied from another student(s), everyone will receive 0 points.

Documentation Requirements

To ensure maintainability, knowledge transfer, and effective use of the system, comprehensive documentation must be prepared and maintained. The documentation should cover the following areas:

Authors documentation

- **Self Evaluation** - Table of the filled out evaluation form (self-evaluation) of what features are implemented. If a feature is implemented, partially lower the number of points in the category accordingly.
- **Authors page** - The authors page must have an image like student ID, passport or driver's licence documents, name and surname, student ID number and email. Email is a link that, when clicked, opens an email client defined in the browser settings. Design the rest of the page as desired.

Technical Documentation

- **System Architecture**: description of the overall architecture, including frontend, backend, and database components, along with integration points.
- **Data Model**: entity-relationship diagrams and explanations of relationships between users, categories etc.
- **API Documentation**: complete specifications of available endpoints, request/response formats, authentication, and error handling.

General Requirements

- Prefer **standard formats** (Markdown, HTML, or PDF) to ensure readability across platforms.
- Documentation should be accessible from the production deployment as a static HTTP content.