User Access Management System

A complete solution for managing user access requests to software applications within an organization. The system supports different roles (Employee, Manager, Admin) with specific permissions for requesting, approving, and managing software access.

Table of Contents

Features

- **Role-based access control** (Employee, Manager, Admin)
- **User authentication** with JWT
- **Software management** (create, view)
- **Access request system** for employees
- **Request review workflow** for managers
- **Responsive frontend interface** using React and Bootstrap
- **RESTful API** with Express.js

- **PostgreSQL database** with TypeORM

Tech Stack

Frontend

- React.js
- React Router for navigation
- React Bootstrap for UI components
- Axios for API calls
- Context API for state management

Backend

- Node.js with Express.js
- PostgreSQL with TypeORM
- JWT for authentication
- bcrypt for password hashing
- dotenv for environment variables

System Architecture

The application follows a client-server architecture:

```
Client (React) <---> Server (Express) <---> Database (PostgreSQL)
```

- **Frontend**: React application with component-based architecture
- **Backend**: Express.js RESTful API with MVC pattern
- **Database**: PostgreSQL relational database managed through TypeORM

Setup Instructions

Prerequisites

- Node.js (v14+ recommended)
- PostgreSQL (v12+ recommended)
- npm or yarn

Backend Setup

1. Clone the repository:

```bash
git clone <repository-url>
cd user-access-management
```

2. Navigate to the backend directory:

```bash
cd backend
```

3. Install dependencies:

```bash
npm install
```

4. Create a `.env` file in the backend directory with the following configuration:

```
PORT=5000
DB_HOST=localhost
DB_PORT=5432
DB_USERNAME=postgres
DB_PASSWORD=your_password
DB_DATABASE=user_access_management
JWT_SECRET=your_jwt_secret_key
```

5. Start the backend server:

```bash
npm run dev
```

Frontend Setup

1. Navigate to the frontend directory:

```bash
cd ../frontend
```

2. Install dependencies:

```bash
npm install
```

3. Start the frontend development server:

```bash
npm start
```

Database Setup

1. Create a PostgreSQL database:

```sql
CREATE DATABASE user_access_management;
```

2. The tables will be automatically created by TypeORM when you start the backend server with the `synchronize: true` option.

API Documentation

Authentication Endpoints

Register a new user

- **URL**: `/api/auth/signup`
- **Method**: `POST`
- **Auth required**: No
- **Request body**:

```json
{
  "username": "string",
  "password": "string",
  "role": "Employee|Manager|Admin"
}
```

- **Success Response**: `201 Created`

```json
{
  "token": "jwt_token",
  "user": {
    "id": "number",
    "username": "string",
    "role": "string"
  }
}
```

Login user

- **URL**: `/api/auth/login`
- **Method**: `POST`
- **Auth required**: No
- **Request body**:

```json
{
  "username": "string",
  "password": "string"
}
```

- **Success Response**: `200 OK`

```json
{
  "token": "jwt_token",
  "user": {
    "id": "number",
    "username": "string",
    "role": "string"
  }
}
```

Software Endpoints

Get all software

- **URL**: `/api/software`
- **Method**: `GET`

- **Auth required**: Yes (Bearer Token)
- **Success Response**: `200 OK`

```json
[
  {
    "id": "number",
    "name": "string",
    "description": "string",
    "accessLevels": ["Read", "Write", "Admin"]
  }
]
```

Create new software

- **URL**: `/api/software`
- **Method**: `POST`
- **Auth required**: Yes (Bearer Token)
- **Permissions**: Admin only
- **Request body**:

```json
{
  "name": "string",
  "description": "string",
  "accessLevels": ["Read", "Write", "Admin"]
}
```

- **Success Response**: `201 Created`

```json
{
  "id": "number",
  "name": "string",
  "description": "string",
  "accessLevels": ["Read", "Write", "Admin"]
}
```

Get software by ID

- **URL**: `/api/software/:id`
- **Method**: `GET`
- **Auth required**: Yes (Bearer Token)
- **Success Response**: `200 OK`

```json
{
  "id": "number",
  "name": "string",
  "description": "string",
  "accessLevels": ["Read", "Write", "Admin"]
}
```

Request Endpoints

Submit access request

- **URL**: `/api/requests`
- **Method**: `POST`
- **Auth required**: Yes (Bearer Token)
- **Permissions**: Employee role
- **Request body**:

```json
{
  "softwareId": "number",
  "accessType": "Read|Write|Admin",
  "reason": "string"
}
```

- **Success Response**: `201 Created`

```json
{
  "id": "number",
  "user": "number",
  "software": "number",
  "accessType": "string",
  "reason": "string",
  "status": "Pending"
}
```

Get pending requests

- **URL**: `/api/requests/pending`
- **Method**: `GET`
- **Auth required**: Yes (Bearer Token)
- **Permissions**: Manager role
- **Success Response**: `200 OK`

```json
[
  {
    "id": "number",
    "user": {
      "id": "number",
      "username": "string",
      "role": "string"
    },
    "software": {
      "id": "number",
      "name": "string",
      "description": "string",
      "accessLevels": ["string"]
    },
    "accessType": "string",
    "reason": "string",
    "status": "Pending"
  }
]
```

Get user requests

- **URL**: `/api/requests/user`
- **Method**: `GET`
- **Auth required**: Yes (Bearer Token)
- **Success Response**: `200 OK`

```json
[
  {
    "id": "number",
    "software": {
      "id": "number",
      "name": "string",
      "description": "string",
      "accessLevels": ["string"]
    },
    "accessType": "string",
    "reason": "string",
    "status": "Pending|Approved|Rejected"
  }
]
```

Update request status

- **URL**: `/api/requests/:id`
- **Method**: `PATCH`
- **Auth required**: Yes (Bearer Token)
- **Permissions**: Manager role
- **Request body**:

```json
{
  "status": "Approved|Rejected"
}
```

- **Success Response**: `200 OK`

```json
{
  "id": "number",
  "user": "number",
  "software": "number",
  "accessType": "string",
  "reason": "string",
  "status": "Approved|Rejected"
}
```

User Roles and Permissions

Employee

- Can submit access requests for software
- Can view their own requests and status

Manager

- Can view pending access requests
- Can approve or reject access requests

Admin

- Has all permissions of Employee and Manager
- Can create new software entries
- Can view all software

Frontend Structure

```
frontend/
├── public/                # Static files
├── src/
│   ├── api/               # API configuration
│   ├── components/        # React components
│   │   ├── admin/         # Admin-specific components
│   │   ├── auth/          # Authentication components
│   │   ├── common/        # Shared UI components
│   │   ├── employee/      # Employee-specific components
│   │   ├── layout/        # Layout components
│   │   └── manager/       # Manager-specific components
│   ├── context/           # Context API providers
│   ├── App.js             # Main component
│   └── index.js           # Entry point
```

Testing

Backend Testing

bash

```bash
# Navigate to backend directory
cd backend

# Run tests
npm test
```

Frontend Testing

bash

```bash
# Navigate to frontend directory
cd frontend

# Run tests
npm test
```

Security Considerations

- JWT-based authentication
- Password hashing with bcrypt
- Role-based access control

- Input validation on both client and server
- CORS configuration for API security
- Environment variables for sensitive information