# LabVisage Documentation

## Overview

LabVisage is a secure and scalable exam monitoring system designed to maintain academic integrity during online lab-based examinations. It provides real-time monitoring of client activities, malpractice detection, and centralized oversight for administrators.

---

## Features

1. **Real-Time Monitoring**: Track all student activity during exams.
2. **Malpractice Detection**: Alerts for suspicious or unauthorized activities.
3. **Secure Exam Sessions**: Lock labs to prevent unauthorized access.
4. **Admin Dashboard**: Manage users, labs, and exams.
5. **User Dashboard**: Start and monitor exams.
6. **Client Registration**: Students can register and participate in exams.
7. **Alerts Dashboard**: View and manage alerts for malpractice.

---

## Project Structure

```
LabVisage/
├── backend/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── app.js
│   ├── Dockerfile
│   ├── package.json
│   └── .env.example
├── frontend/
│   ├── public/
│   ├── src/
│   │   ├── apiService/
│   │   ├── Components/
│   │   ├── Pages/
│   │   ├── Styles/
│   │   ├── App.jsx
│   │   ├── main.jsx
│   ├── Dockerfile
│   ├── vite.config.js
│   ├── package.json
│   ├── index.html
│   ├── .env.example
│   └── .gitignore
├── docker-compose.yml
```

```
├── start.sh
├── start.bat
├── update_ip.sh
└── update_ip.bat
```

---

# Backend

## Technologies

- **Node.js**: Backend runtime.
- **Express.js**: Web framework.
- **MongoDB**: Database for storing users, labs, and exam data.
- **JWT**: Authentication and authorization.

## Key Files

1. `app.js`: Entry point for the backend server.
2. **Controllers**:
   - `userController.js`: Handles user registration, login, and updates.
   - `clientController.js`: Manages client registration, heartbeat, and exam lifecycle.
   - `alertController.js`: Handles alerts for malpractice.
   - `adminController.js`: Admin functionalities like managing labs and users.
3. **Models**:
   - `userModel.js`: Schema for user data.
   - `labModel.js`: Schema for lab data.
4. **Routes**:
   - `userRoutes.js`: Routes for user-related operations.
   - `clientRoutes.js`: Routes for client-related operations.
   - `alertRoutes.js`: Routes for alert-related operations.
   - `adminRoutes.js`: Routes for admin-related operations.
5. **Services**:
   - `clientStore.js`: In-memory store for client data.
   - `alertStore.js`: In-memory store for alerts.
   - `notificationService.js`: Sends notifications.
   - `heartbeatService.js`: Monitors client activity.

## Environment Variables

Refer to `backend/.env.example`:

```
PORT=5000
MONGO_URI=mongodb+srv://<username>:<password>@cluster.mongodb.net/dbname
JWT_SECRET=your_jwt_secret
FRONTEND_URL=http://<Local_IPv4>:3000
```

---

# Frontend

## Technologies Utilized

- **React.js**: Frontend framework.
- **Vite**: Build tool for faster development.
- **Tailwind CSS**: Styling framework.
- **Axios**: HTTP client for API calls.

## Additional Technology

- **Nginx**: Used as a reverse proxy server for serving the frontend application in production.

## Frontend Deployment with Nginx

The frontend application is served using Nginx in the production environment. The `Dockerfile` for the frontend includes the configuration to copy the build files and set up Nginx.

### Nginx Configuration

The Nginx configuration is designed to serve a single-page application (SPA) built with a frontend framework like React, Angular, or Vue.

- **Listening on Port 80**: The server listens for HTTP requests on port 80.
- **Server Name**: The server is configured to respond to requests directed to `localhost`.
- **Root Directory**: The root directory is set to `/usr/share/nginx/html`, where the static files of the frontend application (e.g., `index.html`, CSS, JavaScript) are located.
- **Default File**: The `index.html` file is specified as the default file to serve when accessing the root directory.

#### Routing for SPAs

- The `location /` block uses the `try_files` directive to handle routing:
  - It first checks if the requested file exists (`$uri` or `$uri/`).
  - If the file does not exist, it falls back to serving `index.html`. This ensures that all routes in the SPA are handled by the frontend application, even if they don't correspond to actual files on the server.

#### Custom 404 Page

- The `error_page 404 /index.html;` directive ensures that any 404 errors (e.g., when a user refreshes a non-root route) are redirected to `index.html`, allowing the frontend application to handle the routing.

This configuration is optimized for SPAs deployed in production, ensuring smooth navigation and proper handling of client-side routing.

## Key Files of Frontend

1. `App.jsx`: Main application file defining routes.

2. **Pages**:
    - `Home.jsx`: Landing page.
    - `Login.jsx`: Login page for users.
    - `AdminDashboard.jsx`: Admin dashboard for managing users and labs.
    - `UserDashboard.jsx`: User dashboard for starting and monitoring exams.
    - `ExamMonitoring.jsx`: Real-time monitoring of exams.
    - `AddUser.jsx` & `EditUser.jsx`: Add and edit user details.
    - `AddLab.jsx` & `EditLab.jsx`: Add and edit lab details.
    - `About.jsx`: About page with project details.
3. **Components**:
    - `ProtectedRoute.jsx`: Protects routes based on authentication and roles.
4. **Styles**:
    - `App.css`: Global styles.
    - `index.css`: Tailwind CSS imports.

### Environment Variables of Frontend

Refer to `frontend/.env.example`:

```
VITE_API_URL=http://<your-ip>:5000/api
```

**Note**: `This is required to manually edited during development stage but during build using the scripts automatically updates this link for you.`

---

# Deployment

## Docker

The project uses Docker for containerized deployment.

### Docker Compose

Refer to `docker-compose.yml`:

```
services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    env_file:
      - ./backend/.env

    mem_limit: 512m
    cpus: 0.5

  frontend:
    build:
```

```
      context: ./frontend
      args:
        VITE_API_URL: ${VITE_API_URL}
    ports:
      - "3000:80"
    env_file:
      - ./frontend/.env
    mem_limit: 256m
    cpus: 0.25
    depends_on:
      - backend
```

#### Build and Run

1. Build and start containers:

   ```
   docker-compose up --build
   ```

2. Access the application:
   - Frontend: `http://localhost:3000`
   - Backend: `http://localhost:5000`

---

# Scripts

## Start Scripts (Build Container)

- **Windows**: `start.bat`
- **Linux/Mac**: `start.sh`
- **Note**: Before the build process, ensure the docker engine is running (with Internet Connection). Also, if any possible IPv4 address not found, it's mostly because of Adapter name. In that case, when it asks for IP, input IPv4 address of the main network you are connected to. You can find it checking IPv4 settings of network in Settings application or using command 'ipconfig' in Powershell or Command Prompt for Windows or command 'ip addr' or 'ip a' in shell of Linux Distribution you are using.

## Start Scripts (No Build)

- **Windows**: `start_nobuild.bat`
- **Linux/Mac**: `start_nobuild.sh`

## Update IP Scripts

- **Windows**: `update_ip.bat`
- **Linux/Mac**: `update_ip.sh`

---

# API Endpoints

### User Routes

- `POST /api/users/register`: Register a new user (Admin only).
- `POST /api/users/login`: Login a user.
- `GET /api/users/:userId`: Get user details.
- `POST /api/users/:userId`: Edit user details.

### Client Routes

- `POST /api/clients/register`: Register a client.
- `POST /api/clients/heartbeat`: Send client heartbeat.
- `POST /api/clients/createexam`: Create a new exam.
- `POST /api/clients/startexam`: Start an exam.
- `POST /api/clients/endexam`: End an exam.
- `GET /api/clients/getallclients/:labcode`: Get all clients in a lab.

### Alert Routes

- `POST /api/alerts/:labCode/:clientId`: Report an alert.
- `GET /api/alerts/:labCode`: Get alerts for a lab.

### Admin Routes

- `POST /api/admin/labs`: Add a new lab.
- `DELETE /api/admin/labs/:labCode`: Delete a lab.
- `POST /api/admin/labs/:labId`: Edit a lab.
- `GET /api/admin/labs`: Get all labs.
- `GET /api/admin/labs/:labId`: Get lab details.
- `GET /api/admin/users`: Get all users.
- `DELETE /api/admin/users/:userId`: Delete a user.

---

## Development

### For Frontend

1. Install dependencies:

```
npm install
```

2. Start the development server:

```
npm run dev
```

### For Backend

1. Install dependencies:

```
npm install
```

2. Start the development server:

```
npm run dev
```

---

## Contributors

| USN | Name | Contribution |
|---|---|---|
| 2KE21CS100 | Shrinivas Masti | Testing & Documentation |
| 2KE21CS102 | Shripad Kulkarni | Frontend Development |
| 2KE21CS109 | Srinidhi Chappar | Client Monitoring System |
| 2KE21CS113 | Suraj Kr Das | Backend API Development |

---

## Acknowledgments


- **Project Guide**: Mrs. Suman Yaligar
    - For her invaluable guidance and support throughout the project.


---

## License