Integrative Programming and Technologies ~ Final

Full-Stack Application

Project Description

A full-stack application for user authentication, account management, employee management with dynamic departments, transactional workflows, and employee requests. Built using Node.js + MySQL for the backend and Angular 17/19 for the frontend. Key features include email-based user registration with verification, JWT-based authentication and authorization, and role-based access control (Admin and User roles).

Features Overview

Accounts & Authentication

- **Email Sign-Up**: Users can register with an email and password, receiving a verification email to activate their account.
- **Verify Email**: Email verification link sent via SMTP (using Ethereal for testing) to confirm user registration.
- **Authentication**: JWT-based authentication with access and refresh tokens.
- **Authorization**: Role-based access control (Admin and User roles). Admins have elevated privileges (e.g., managing employees, departments, workflows, requests).
- **Account Management**: Users can view and update their profiles, change passwords, and manage refresh tokens.

Employees

- **Employee Management**: Create, read, update, and delete (CRUD) employees with details like Employee ID, **Account** assignment, position, **Department**, hire date, and status.
- **Department Transfer**: Admins can transfer employees between departments, creating a workflow entry for tracking.

Departments

- **Department Management**: CRUD operations for departments with fields for name and description.
- **Employee Assignment**: Departments are linked to employees, showing employee counts per department.

Workflows

- **Transactional Workflows**: Create workflows for employee-related actions (e.g., onboarding, department changes, employee requests).
- **Status Management**: Update workflow status (e.g., Pending, Approved, Rejected) with detailed tracking.

Requests

- **Employee Requests**: Employees can create requests (e.g., equipment, leave, resources) with a header (type, status) and a list of items (e.g., name, quantity).
- **Request Management**: Admins can view, update, and delete requests. Users can view their own requests.

Setup Instructions

Prerequisites

- Node.js (v16+)
- MySQL Community Server
- Angular CLI (v19+)
- Postman (for API testing)

Backend Setup

1. Clone the repository: git clone <repository-url>
2. Navigate to the backend folder: cd backend
3. Install dependencies: npm install
4. Configure MySQL and SMTP settings in config.json:

```
{
  "database": {
    "host": "localhost",
    "port": 3306,
    "user": "root",
    "password": "your-password",
    "database": "fullstack_db"
  },
  "secret": "your-random-secret",
  "emailFrom": "no-reply@yourapp.com",
  "smtpOptions": {
    "host": "smtp.ethereal.email",
    "port": 587,
    "auth": {
      "user": "your-ethereal-user",
      "pass": "your-ethereal-pass"
    }
  }
}
```

5. Start the backend: npm start

Frontend Setup

1. Navigate to the frontend folder: cd frontend
2. Install dependencies: npm install
3. Start the frontend: ng serve

4. To use the fake backend, ensure FakeBackendInterceptor is included in app.module.ts. To switch to the real backend, remove it and update environment.apiUrl to http://localhost:4000.

Code Implementation

Backend API (Node.js + MySQL)

Employees

employees/index.js

```
const express = require('express');
const router = express.Router();
const db = require('../_helpers/db');
const authorize = require('../_middleware/authorize');
const Role = require('../_helpers/role');

router.post('/', authorize(Role.Admin), create);
router.get('/', authorize(), getAll);
router.get('/:id', authorize(), getById);
router.put('/:id', authorize(Role.Admin), update);
router.delete('/:id', authorize(Role.Admin), _delete);
router.post('/:id/transfer', authorize(Role.Admin), transfer);

async function create(req, res, next) {
    try {
        const employee = await db.Employee.create(req.body);
        res.status(201).json(employee);
    } catch (err) { next(err); }
}

async function getAll(req, res, next) {
    try {
        const employees = await db.Employee.findAll({
            include: [{ model: db.User }, { model: db.Department }]
        });
        res.json(employees);
    } catch (err) { next(err); }
}
```

```
async function getById(req, res, next) {
  try {
    const employee = await db.Employee.findByPk(req.params.id, {
      include: [{ model: db.User }, { model: db.Department }]
    });
    if (!employee) throw new Error('Employee not found');
    res.json(employee);
  } catch (err) { next(err); }
}

async function update(req, res, next) {
  try {
    const employee = await db.Employee.findByPk(req.params.id);
    if (!employee) throw new Error('Employee not found');
    await employee.update(req.body);
    res.json(employee);
  } catch (err) { next(err); }
}

async function _delete(req, res, next) {
  try {
    const employee = await db.Employee.findByPk(req.params.id);
    if (!employee) throw new Error('Employee not found');
    await employee.destroy();
    res.json({ message: 'Employee deleted' });
  } catch (err) { next(err); }
}
```

```
async function transfer(req, res, next) {
  try {
    const employee = await db.Employee.findByPk(req.params.id);
    if (!employee) throw new Error('Employee not found');
    await employee.update({ departmentId: req.body.departmentId });
    await db.Workflow.create({
      employeeId: employee.id,
      type: 'Transfer',
      details: { newDepartmentId: req.body.departmentId }
    });
    res.json({ message: 'Employee transferred' });
  } catch (err) { next(err); }
}

module.exports = router;
```

Departments

departments/index.js

```
const express = require('express');
const router = express.Router();
const db = require('../_helpers/db');
const authorize = require('../_middleware/authorize');
const Role = require('../_helpers/role');

router.post('/', authorize(Role.Admin), create);
router.get('/', authorize(), getAll);
router.get('/:id', authorize(), getById);
router.put('/:id', authorize(Role.Admin), update);
router.delete('/:id', authorize(Role.Admin), _delete);

async function create(req, res, next) {
    try {
        const department = await db.Department.create(req.body);
        res.status(201).json(department);
    } catch (err) { next(err); }
}

async function getAll(req, res, next) {
    try {
        const departments = await db.Department.findAll({
            include: [{ model: db.Employee, attributes: ['id'] }]
        });
        res.json(departments.map(d => ({
            ...d.toJSON(),
            employeeCount: d.Employees.length
        })));
    } catch (err) { next(err); }
}
```

```
async function getById(req, res, next) {
  try {
    const department = await db.Department.findByPk(req.params.id, {
      include: [{ model: db.Employee, attributes: ['id'] }]
    });
    if (!department) throw new Error('Department not found');
    res.json({ ...department.toJSON(), employeeCount: department.Employees.length });
  } catch (err) { next(err); }
}

async function update(req, res, next) {
  try {
    const department = await db.Department.findByPk(req.params.id);
    if (!department) throw new Error('Department not found');
    await department.update(req.body);
    res.json(department);
  } catch (err) { next(err); }
}

async function _delete(req, res, next) {
  try {
    const department = await db.Department.findByPk(req.params.id);
    if (!department) throw new Error('Department not found');
    await department.destroy();
    res.json({ message: 'Department deleted' });
  } catch (err) { next(err); }
}

module.exports = router;
```

Workflows

workflows/index.js

```
const express = require('express');
const router = express.Router();
const db = require('../_helpers/db');
const authorize = require('../_middleware/authorize');
const Role = require('../_helpers/role');

router.post('/', authorize(Role.Admin), create);
router.get('/employee/:employeeId', authorize(), getByEmployeeId);
router.put('/:id/status', authorize(Role.Admin), updateStatus);
router.post('/onboarding', authorize(Role.Admin), onboarding);

async function create(req, res, next) {
    try {
        const workflow = await db.Workflow.create(req.body);
        res.status(201).json(workflow);
    } catch (err) { next(err); }
}

async function getByEmployeeId(req, res, next) {
    try {
        const workflows = await db.Workflow.findAll({
            where: { employeeId: req.params.employeeId }
        });
        res.json(workflows);
    } catch (err) { next(err); }
}
```

```
async function updateStatus(req, res, next) {
  try {
    const workflow = await db.Workflow.findByPk(req.params.id);
    if (!workflow) throw new Error('Workflow not found');
    await workflow.update({ status: req.body.status });
    res.json(workflow);
  } catch (err) { next(err); }
}

async function onboarding(req, res, next) {
  try {
    const workflow = await db.Workflow.create({
      employeeId: req.body.employeeId,
      type: 'Onboarding',
      details: req.body.details,
      status: 'Pending'
    });
    res.status(201).json(workflow);
  } catch (err) { next(err); }
}

module.exports = router;
```

Requests

requests/index.js

```
const express = require('express');
const router = express.Router();
const db = require('../_helpers/db');
const authorize = require('../_middleware/authorize');
const Role = require('../_helpers/role');

router.post('/', authorize(), create);
router.get('/', authorize(Role.Admin), getAll);
router.get('/:id', authorize(), getById);
router.get('/employee/:employeeId', authorize(), getByEmployeeId);
router.put('/:id', authorize(Role.Admin), update);
router.delete('/:id', authorize(Role.Admin), _delete);

async function create(req, res, next) {
    try {
        const request = await db.Request.create({
            ...req.body,
            employeeId: req.user.employeeId
        }, {
            include: [{ model: db.RequestItem }]
        });
        res.status(201).json(request);
    } catch (err) { next(err); }
}
```

```
async function getAll(req, res, next) {
  try {
    const requests = await db.Request.findAll({
      include: [{ model: db.RequestItem }, { model: db.Employee }]
    });
    res.json(requests);
  } catch (err) { next(err); }
}

async function getById(req, res, next) {
  try {
    const request = await db.Request.findByPk(req.params.id, {
      include: [{ model: db.RequestItem }, { model: db.Employee }]
    });
    if (!request) throw new Error('Request not found');
    if (req.user.role !== Role.Admin && request.employeeId !== req.user.employeeId) {
      throw new Error('Unauthorized');
    }
    res.json(request);
  } catch (err) { next(err); }
}

async function getByEmployeeId(req, res, next) {
  try {
    const requests = await db.Request.findAll({
      where: { employeeId: req.params.employeeId },
      include: [{ model: db.RequestItem }]
    });
    res.json(requests);
  } catch (err) { next(err); }
}
```

```
async function update(req, res, next) {
  try {
    const request = await db.Request.findByPk(req.params.id);
    if (!request) throw new Error('Request not found');
    await request.update(req.body);
    if (req.body.items) {
      await db.RequestItem.destroy({ where: { requestId: request.id } });
      await db.RequestItem.bulkCreate(req.body.items.map(item => ({
        ...item,
        requestId: request.id
      })));
    }
    res.json(request);
  } catch (err) { next(err); }
}

async function _delete(req, res, next) {
  try {
    const request = await db.Request.findByPk(req.params.id);
    if (!request) throw new Error('Request not found');
    await request.destroy();
    res.json({ message: 'Request deleted' });
  } catch (err) { next(err); }
}

module.exports = router;
```

Frontend (Angular)

Employees

employees/list.component.html

```
<div class="card">
  <div class="card-header">Employees</div>
  <div class="card-body">
    <div class="table-responsive">
      <table class="table table-striped">
        <thead>
          <tr>
            <th>Employee ID</th>
            <th>User</th>
            <th>Position</th>
            <th>Department</th>
            <th>Hire Date</th>
            <th>Status</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let employee of employees">
            <td>{{employee.employeeId}}</td>
            <td>{{employee.user?.email}}</td>
            <td>{{employee.position}}</td>
            <td>{{employee.department?.name}}</td>
            <td>{{employee.hireDate | date:'shortDate'}}</td>
            <td><span class="badge" [ngClass]="{'bg-success': employee.status === 'Active', 'bg-danger': employee.statu
s !== 'Active'}">{{employee.status}}</span></td>
```

```
            <td>
              <button class="btn btn-sm btn-info me-1" (click)="viewRequests(employee.id)">Requests</button>
              <button class="btn btn-sm btn-info me-1" (click)="viewWorkflows(employee.id)">Workflows</button>
              <button *ngIf="account()?.role === 'Admin'" class="btn btn-sm btn-warning me-1" (click)="transfer(employe
e)">Transfer</button>
              <button class="btn btn-sm btn-primary me-1" (click)="edit(employee.id)">Edit</button>
              <button *ngIf="account()?.role === 'Admin'" class="btn btn-sm btn-danger" (click)="delete(employee.id)">De
lete</button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
    <button *ngIf="account()?.role === 'Admin'" class="btn btn-primary float-end" (click)="add()">Add Employee</button>
  </div>
</div>
```

employees/add-edit.component.html

```
<div class="card">
  <div class="card-header">{{id ? 'Edit' : 'Add'}} Employee</div>
  <div class="card-body">
    <div class="alert alert-danger" *ngIf="errorMessage">{{errorMessage}}</div>
    <div class="mb-3">
      <label class="form-label">Employee ID</label>
      <input type="text" class="form-control" [(ngModel)]="employee.employeeId" [disabled]="!!id">
    </div>
    <div class="mb-3">
      <label class="form-label">User</label>
      <select class="form-select" [(ngModel)]="employee.userId">
        <option *ngFor="let user of users" [value]="user.id">{{user.email}}</option>
      </select>
    </div>
    <div class="mb-3">
      <label class="form-label">Position</label>
      <input type="text" class="form-control" [(ngModel)]="employee.position">
    </div>
    <div class="mb-3">
      <label class="form-label">Department</label>
      <select class="form-select" [(ngModel)]="employee.departmentId">
        <option *ngFor="let dept of departments" [value]="dept.id">{{dept.name}}</option>
      </select>
    </div>
    <div class="mb-3">
      <label class="form-label">Hire Date</label>
      <input type="date" class="form-control" [(ngModel)]="employee.hireDate">
    </div>
```

```
      <div class="mb-3">
        <label class="form-label">Status</label>
        <select class="form-select" [(ngModel)]="employee.status">
          <option>Active</option>
          <option>Inactive</option>
        </select>
      </div>
      <div class="text-center">
        <button class="btn btn-primary me-2" (click)="save()">Save</button>
        <button class="btn btn-secondary" (click)="cancel()">Cancel</button>
      </div>
    </div>
  </div>
```

employees/transfer.component.html

```
<div class="modal fade show d-block" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Transfer Employee: {{employee?.employeeId}}</h5>
        <button type="button" class="btn-close" (click)="cancel()"></button>
      </div>
      <div class="modal-body">
        <div class="mb-3">
          <label class="form-label">Department</label>
          <select class="form-select" [(ngModel)]="departmentId">
            <option *ngFor="let dept of departments" [value]="dept.id">{{dept.name}}</option>
          </select>
        </div>
      </div>
      <div class="modal-footer">
        <button class="btn btn-warning" (click)="transfer()">Transfer</button>
        <button class="btn btn-secondary" (click)="cancel()">Cancel</button>
      </div>
    </div>
  </div>
</div>
<div class="modal-backdrop fade show"></div>
```

Departments

departments/list.component.html

```
<div class="card">
  <div class="card-header">Departments</div>
  <div class="card-body">
    <div class="table-responsive">
      <table class="table table-striped">
        <thead>
          <tr>
            <th>Name</th>
            <th>Description</th>
            <th>Employee Count</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let dept of departments">
            <td>{{dept.name}}</td>
            <td>{{dept.description}}</td>
            <td>{{dept.employeeCount}}</td>
            <td>
              <button class="btn btn-sm btn-primary me-1" (click)="edit(dept.id)">Edit</button>
              <button *ngIf="account()?.role === 'Admin'" class="btn btn-sm btn-danger" (click)="delete(dept.id)">Delete
</button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
    <button *ngIf="account()?.role === 'Admin'" class="btn btn-primary float-end" (click)="add()">Add Department</button>
  </div>
</div>
```

departments/add-edit.component.html

```
<div class="card">
  <div class="card-header">{{id ? 'Edit' : 'Add'}} Department</div>
  <div class="card-body">
    <div class="alert alert-danger" *ngIf="errorMessage">{{errorMessage}}</div>
    <div class="mb-3">
      <label class="form-label">Name</label>
      <input type="text" class="form-control" [(ngModel)]="department.name">
    </div>
    <div class="mb-3">
      <label class="form-label">Description</label>
      <input type="text" class="form-control" [(ngModel)]="department.description">
    </div>
    <div class="text-center">
      <button class="btn btn-primary me-2" (click)="save()">Save</button>
      <button class="btn btn-secondary" (click)="cancel()">Cancel</button>
    </div>
  </div>
</div>
```

Workflows

workflows/list.component.html

```
<div class="card">
  <div class="card-header">Workflows for Employee {{employeeId}}</div>
  <div class="card-body">
    <div class="table-responsive">
      <table class="table table-striped">
        <thead>
          <tr>
            <th>Type</th>
            <th>Details</th>
            <th>Status</th>
            <th *ngIf="account()?.role === 'Admin'">Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let workflow of workflows">
            <td>{{workflow.type}}</td>
            <td>{{workflow.details | json}}</td>
            <td>{{workflow.status}}</td>
            <td *ngIf="account()?.role === 'Admin'">
              <select class="form-select d-inline-block w-auto" [(ngModel)]="workflow.status" (change)="updateStatus(w
orkflow)">
                <option>Pending</option>
                <option>Approved</option>
                <option>Rejected</option>
              </select>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>
```

Requests

requests/list.component.html

```
<div class="card">
  <div class="card-header">Requests</div>
  <div class="card-body">
    <div class="table-responsive">
      <table class="table table-striped">
        <thead>
          <tr>
            <th>Type</th>
            <th>Employee</th>
            <th>Items</th>
            <th>Status</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let request of requests">
            <td>{{request.type}}</td>
            <td>{{request.employee?.employeeId}}</td>
            <td>
              <ul>
                <li *ngFor="let item of request.requestItems">{{item.name}} (x{{item.quantity}})</li>
              </ul>
            </td>
            <td>{{request.status}}</td>
            <td>
              <button *nglf="account()?.role === 'Admin'" class="btn btn-sm btn-primary me-1" (click)="edit(request.id)">
Edit</button>
              <button *nglf="account()?.role === 'Admin'" class="btn btn-sm btn-danger" (click)="delete(request.id)">Dele
te</button>
            </td>
          </tr>
        </tbody>
      </table>
```

```
    </div>
    <button class="btn btn-primary float-end" (click)="add()">Add Request</button>
  </div>
</div>
```

requests/add-edit.component.html

```
<div class="card">
  <div class="card-header">{{id ? 'Edit' : 'Add'}} Request</div>
  <div class="card-body">
    <div class="alert alert-danger" *ngIf="errorMessage">{{errorMessage}}</div>
    <div class="mb-3">
      <label class="form-label">Type</label>
      <select class="form-select" [(ngModel)]="request.type">
        <option>Equipment</option>
        <option>Leave</option>
        <option>Resources</option>
      </select>
    </div>
    <div class="mb-3">
      <label class="form-label">Items</label>
      <div *ngFor="let item of request.items; let i = index" class="border p-2 mb-2">
        <div class="row">
          <div class="col-md-5">
            <label class="form-label">Name</label>
            <input type="text" class="form-control" [(ngModel)]="item.name">
          </div>
          <div class="col-md-5">
            <label class="form-label">Quantity</label>
            <input type="number" class="form-control" [(ngModel)]="item.quantity">
          </div>
          <div class="col-md-2 d-flex align-items-end">
            <button class="btn btn-danger" (click)="removeItem(i)">Remove</button>
          </div>
        </div>
      </div>
      <button class="btn btn-secondary" (click)="addItem()">Add Item</button>
    </div>
```

```
      <div class="text-center">
        <button class="btn btn-primary me-2" (click)="save()">Save</button>
        <button class="btn btn-secondary" (click)="cancel()">Cancel</button>
      </div>
    </div>
  </div>
```

Fake Backend Provider (Angular)

fake-backend.ts

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpResponse, HttpHandler, HttpEvent, HttpInterceptor, HTTP_INTERCEPTORS } from '@angular/commo
n/http';
import { Observable, of, throwError } from 'rxjs';
import { delay, mergeMap, materialize, dematerialize } from 'rxjs/operators';

@Injectable()
export class FakeBackendInterceptor implements HttpInterceptor {
  private users = [
    { id: 1, email: 'admin@example.com', password: 'admin', role: 'Admin', employeeId: 1 },
    { id: 2, email: 'user@example.com', password: 'user', role: 'User', employeeId: 2 }
  ];
  private employees = [
    { id: 1, employeeId: 'EMP001', userId: 1, position: 'Developer', departmentId: 1, hireDate: '2025-01-01', status: 'Active' },
    { id: 2, employeeId: 'EMP002', userId: 2, position: 'Designer', departmentId: 2, hireDate: '2025-02-01', status: 'Active' }
  ];
  private departments = [
    { id: 1, name: 'Engineering', description: 'Software development team', employeeCount: 1 },
    { id: 2, name: 'Marketing', description: 'Marketing team', employeeCount: 1 }
  ];
  private workflows = [
    { id: 1, employeeId: 1, type: 'Onboarding', details: { task: 'Setup workstation' }, status: 'Pending' }
  ];
  private requests = [
    { id: 1, employeeId: 2, type: 'Equipment', requestItems: [{ name: 'Laptop', quantity: 1 }], status: 'Pending' }
  ];
```

```
intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const { url, method, headers, body } = request;

    return of(null)
        .pipe(mergeMap(() => this.handleRoute(url, method, headers, body)))
        .pipe(materialize())
        .pipe(delay(500))
        .pipe(dematerialize());
}

private handleRoute(url: string, method: string, headers: any, body: any): Observable<HttpEvent<any>> {
    // Accounts Routes
    if (url.endsWith('/accounts/authenticate') && method === 'POST') {
        const { email, password } = body;
        const user = this.users.find(u => u.email === email && u.password === password);
        if (!user) return throwError(() => new Error('Invalid credentials'));
        return of(new HttpResponse({ status: 200, body: { ...user, token: 'fake-jwt-token' } }));
    }

    if (url.endsWith('/accounts') && method === 'GET') {
        return this.authorize(headers, 'Admin', () => of(new HttpResponse({ status: 200, body: this.users })));
    }

    // Employees Routes
    if (url.endsWith('/employees') && method === 'GET') {
        return this.authorize(headers, null, () => of(new HttpResponse({ status: 200, body: this.employees })));
    }
```

```
    if (url.endsWith('/employees') && method === 'POST') {
        return this.authorize(headers, 'Admin', () => {
            const employee = { id: this.employees.length + 1, ...body };
            this.employees.push(employee);
            return of(new HttpResponse({ status: 201, body: employee }));
        });
    }

    if (url.match(/\/employees\/\d+$/) && method === 'GET') {
        const id = parseInt(url.split('/').pop());
        const employee = this.employees.find(e => e.id === id);
        return this.authorize(headers, null, () => employee ? of(new HttpResponse({ status: 200, body: employee })) : throwErro
r(() => new Error('Employee not found')));
    }

    if (url.match(/\/employees\/\d+$/) && method === 'PUT') {
        return this.authorize(headers, 'Admin', () => {
            const id = parseInt(url.split('/').pop());
            const employeeIndex = this.employees.findIndex(e => e.id === id);
            if (employeeIndex === -1) return throwError(() => new Error('Employee not found'));
            this.employees[employeeIndex] = { id, ...body };
            return of(new HttpResponse({ status: 200, body: this.employees[employeeIndex] }));
        });
    }

    if (url.match(/\/employees\/\d+$/) && method === 'DELETE') {
        return this.authorize(headers, 'Admin', () => {
            const id = parseInt(url.split('/').pop());
            this.employees = this.employees.filter(e => e.id !== id);
            return of(new HttpResponse({ status: 200, body: { message: 'Employee deleted' } }));
        });
    }
```

```
if (url.match(/\/employees\/\d+\/transfer$/) && method === 'POST') {
    return this.authorize(headers, 'Admin', () => {
        const id = parseInt(url.split('/')[2]);
        const employee = this.employees.find(e => e.id === id);
        if (!employee) return throwError(() => new Error('Employee not found'));
        employee.departmentId = body.departmentId;
        this.workflows.push({ id: this.workflows.length + 1, employeeId: id, type: 'Transfer', details: body, status: 'Pending' });
        return of(new HttpResponse({ status: 200, body: { message: 'Employee transferred' } }));
    });
}

// Departments Routes
if (url.endsWith('/departments') && method === 'GET') {
    return this.authorize(headers, null, () => of(new HttpResponse({ status: 200, body: this.departments })));
}

if (url.endsWith('/departments') && method === 'POST') {
    return this.authorize(headers, 'Admin', () => {
        const department = { id: this.departments.length + 1, ...body, employeeCount: 0 };
        this.departments.push(department);
        return of(new HttpResponse({ status: 201, body: department }));
    });
}

if (url.match(/\/departments\/\d+$/) && method === 'PUT') {
    return this.authorize(headers, 'Admin', () => {
        const id = parseInt(url.split('/').pop());
        const deptIndex = this.departments.findIndex(d => d.id === id);
        if (deptIndex === -1) return throwError(() => new Error('Department not found'));
        this.departments[deptIndex] = { id, ...body, employeeCount: this.departments[deptIndex].employeeCount };
        return of(new HttpResponse({ status: 200, body: this.departments[deptIndex] }));
    });
}
```

```
if (url.match(/\/departments\/\d+$/) && method === 'DELETE') {
    return this.authorize(headers, 'Admin', () => {
        const id = parseInt(url.split('/').pop());
        this.departments = this.departments.filter(d => d.id !== id);
        return of(new HttpResponse({ status: 200, body: { message: 'Department deleted' } }));
    });
}

// Workflows Routes
if (url.match(/\/workflows\/employee\/\d+$/) && method === 'GET') {
    return this.authorize(headers, null, () => {
        const employeeId = parseInt(url.split('/').pop());
        const workflows = this.workflows.filter(w => w.employeeId === employeeId);
        return of(new HttpResponse({ status: 200, body: workflows }));
    });
}

if (url.endsWith('/workflows') && method === 'POST') {
    return this.authorize(headers, 'Admin', () => {
        const workflow = { id: this.workflows.length + 1, ...body };
        this.workflows.push(workflow);
        return of(new HttpResponse({ status: 201, body: workflow }));
    });
}

// Requests Routes
if (url.endsWith('/requests') && method === 'GET') {
    return this.authorize(headers, 'Admin', () => of(new HttpResponse({ status: 200, body: this.requests })));
}
```

```
    if (url.endsWith('/requests') && method === 'POST') {
        return this.authorize(headers, null, () => {
            const request = { id: this.requests.length + 1, employeeId: this.getUser(headers).employeeId, ...body };
            this.requests.push(request);
            return of(new HttpResponse({ status: 201, body: request }));
        });
    }

    if (url.match(/\/requests\/\d+$/) && method === 'PUT') {
        return this.authorize(headers, 'Admin', () => {
            const id = parseInt(url.split('/').pop());
            const reqIndex = this.requests.findIndex(r => r.id === id);
            if (reqIndex === -1) return throwError(() => new Error('Request not found'));
            this.requests[reqIndex] = { id, ...body };
            return of(new HttpResponse({ status: 200, body: this.requests[reqIndex] }));
        });
    }

    if (url.match(/\/requests\/\d+$/) && method === 'DELETE') {
        return this.authorize(headers, 'Admin', () => {
            const id = parseInt(url.split('/').pop());
            this.requests = this.requests.filter(r => r.id !== id);
            return of(new HttpResponse({ status: 200, body: { message: 'Request deleted' } }));
        });
    }

    return next.handle(request);
}
```

```
    private authorize(headers: any, requiredRole: string | null, success: () => Observable<HttpEvent<any>>): Observable<HttpE
vent<any>> {
        const user = this.getUser(headers);
        if (!user) return throwError(() => new Error('Unauthorized'));
        if (requiredRole && user.role !== requiredRole) return throwError(() => new Error('Forbidden'));
        return success();
    }

    private getUser(headers: any) {
        const authHeader = headers.get('Authorization');
        if (!authHeader || authHeader !== 'Bearer fake-jwt-token') return null;
        return this.users.find(u => u.token === 'fake-jwt-token');
    }
}

export const fakeBackendProvider = {
    provide: HTTP_INTERCEPTORS,
    useClass: FakeBackendInterceptor,
    multi: true
};
```

API Endpoints

Accounts

Method	Endpoint	Description	Authentication
POST	/accounts/register	Register a new user	None
POST	/accounts/verify-email	Verify email with token	None
POST	/accounts/authenticate	Authenticate user and get JWT tokens	None
POST	/accounts/refresh-token	Refresh JWT token	None
POST	/accounts/revoke-token	Revoke refresh token	Authenticated
GET	/accounts	Get all accounts	Admin
GET	/accounts/:id	Get account by ID	Authenticated
PUT	/accounts/:id	Update account	Authenticated

Example Request (POST /accounts/register):

{

```
  "firstName": "John",
  "lastName": "Doe",
  "email": "john.doe@example.com",
  "password": "Password123!"
}
```

Employees

Method	Endpoint	Description	Authentication
POST	/employees	Create a new employee	Admin
GET	/employees	Get all employees	Authenticated
GET	/employees/:id	Get employee by ID	Authenticated
PUT	/employees/:id	Update employee	Admin
DELETE	/employees/:id	Delete employee	Admin
POST	/employees/:id/transfer	Transfer employee to a new department	Admin

Example Request (POST /employees):

```
{
  "employeeId": "EMP001",
  "userId": 1,
  "position": "Developer",
  "hireDate": "2025-01-01",
  "departmentId": 1
}
```

Departments

Method	Endpoint	Description	Authentication
POST	/departments	Create a new department	Admin
GET	/departments	Get all departments	Authenticated
GET	/departments/:id	Get department by ID	Authenticated
PUT	/departments/:id	Update department	Admin
DELETE	/departments/:id	Delete department	Admin

Example Request (POST /departments):

```
{
  "name": "Engineering",
  "description": "Software development team"
}
```

Workflows

Method	Endpoint	Description	Authentication
POST	/workflows	Create a new workflow	Admin
GET	/workflows/employee/:employeeId	Get workflows for an employee	Authenticated
PUT	/workflows/:id/status	Update workflow status	Admin
POST	/workflows/onboarding	Initiate employee onboarding	Admin

Example Request (POST /workflows):

```
{
  "employeeId": 1,
  "type": "Onboarding",
  "details": { "task": "Setup workstation" }
}
```

Requests

Method	Endpoint	Description	Authentication
POST	/requests	Create a new request	Authenticated
GET	/requests	Get all requests	Admin
GET	/requests/:id	Get request by ID	Authenticated
GET	/requests/employee/:employeeId	Get requests for an employee	Authenticated
PUT	/requests/:id	Update request	Admin
DELETE	/requests/:id	Delete request	Admin

Example Request (POST /requests):

```
{
  "employeeId": 1,
  "type": "Equipment",
  "items": [
    { "name": "Laptop", "quantity": 1 },
    { "name": "Monitor", "quantity": 2 }
  ]
}
```

Frontend Features

- **User Authentication**: Email sign-up with verification, login/logout, and profile management.
- **Role-Based Access**: Admin users have access to additional features (e.g., employee management, department management).
- **Employee Management**: View, add, edit, transfer, and delete employees with user and department assignments.
- **Department Management**: Manage dynamic departments as a settings module.
- **Workflow Management**: Create and manage transactional workflows (e.g., onboarding, department changes).
- **Request Management**: Create and manage employee requests with multiple items (e.g., equipment, leave).
- **Responsive UI**: Bootstrap-based tables and forms with validation and error handling.
- **Fake Backend**: Simulates API responses for development without a real backend.

Testing Instructions

Backend

1. Use Postman to test API endpoints (e.g., POST http://localhost:4000/accounts/register).
2. Run unit tests: npm test (requires test setup in package.json).

Frontend

1. Test with fake backend: Ensure FakeBackendInterceptor is in app.module.ts and run ng serve.
2. Test with real backend: Remove FakeBackendInterceptor, update environment.apiUrl, and run ng serve.
3. Run unit tests: ng test.