Student Activities - Backend Code (Node.js + Express + MongoDB)

Project Structure

```
backend/
├── config/
│   └── db.js
├── middleware/
│   ├── auth.js
│   └── upload.js
├── models/
│   ├── User.js
│   ├── Notice.js
│   ├── Department.js
│   └── Club.js
├── routes/
│   ├── auth.js
│   ├── users.js
│   ├── notices.js
│   ├── departments.js
│   └── clubs.js
├── uploads/
├── .env
├── server.js
└── package.json
```

1. package.json

```
{
  "name": "student-activities-backend",
  "version": "1.0.0",
  "description": "Backend for Student Activities Notice Board",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^8.0.0",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.2",
    "dotenv": "^16.3.1",
    "cors": "^2.8.5",
    "multer": "^1.4.5-lts.1",
    "express-validator": "^7.0.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

2. .env

```
PORT=5000
MONGODB_URI=mongodb://localhost:27017/student_activities
JWT_SECRET=your_super_secret_jwt_key_change_this_in_production
JWT_EXPIRE=7d
```

3. server.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');
const path = require('path');

// Load env vars
dotenv.config();

// Import routes
const authRoutes = require('./routes/auth');
const userRoutes = require('./routes/users');
const noticeRoutes = require('./routes/notices');
const departmentRoutes = require('./routes/departments');
const clubRoutes = require('./routes/clubs');

// Initialize express app
const app = express();

// Middleware
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Serve static files (uploaded PDFs)
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

// Connect to MongoDB
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() =&gt; console.log('MongoDB connected successfully'))
.catch((err) =&gt; console.error('MongoDB connection error:', err));

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/users', userRoutes);
app.use('/api/notices', noticeRoutes);
app.use('/api/departments', departmentRoutes);
app.use('/api/clubs', clubRoutes);

// Health check route
app.get('/api/health', (req, res) =&gt; {
  res.json({ status: 'OK', message: 'Server is running' });
});

// Error handling middleware
app.use((err, req, res, next) =&gt; {
  console.error(err.stack);
  res.status(err.status || 500).json({
    success: false,
    message: err.message || 'Internal Server Error',
  });
});

// Start server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () =&gt; {
  console.log(`Server running on port ${PORT}`);
});
```

4. models/User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please provide a name'],
    trim: true,
  },
  email: {
    type: String,
    required: [true, 'Please provide an email'],
    unique: true,
    lowercase: true,
    match: [
      /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/,
      'Please provide a valid email',
    ],
  },
  password: {
    type: String,
    required: [true, 'Please provide a password'],
    minlength: 6,
    select: false,
  },
  role: {
    type: String,
    enum: ['student', 'faculty', 'club_member', 'admin'],
    default: 'student',
  },
  department: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Department',
  },
  club: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Club',
  },
  canPost: {
    type: Boolean,
    default: false,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
});

// Hash password before saving
userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) {
    next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

// Method to compare passwords
userSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

module.exports = mongoose.model('User', userSchema);
```

5. models/Notice.js

```
const mongoose = require('mongoose');

const noticeSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Please provide a notice title'],
    trim: true,
  },
  content: {
    type: String,
    trim: true,
  },
  type: {
    type: String,
    enum: ['academic', 'club'],
    required: true,
  },
  format: {
    type: String,
    enum: ['text', 'pdf'],
    required: true,
  },
  pdfFile: {
    filename: String,
    path: String,
    size: Number,
  },
  department: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Department',
  },
  club: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Club',
  },
  postedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  isActive: {
    type: Boolean,
    default: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
  updatedAt: {
    type: Date,
    default: Date.now,
  },
});

// Update updatedAt on save
noticeSchema.pre('save', function (next) {
  this.updatedAt = Date.now();
  next();
});

module.exports = mongoose.model('Notice', noticeSchema);
```

6. models/Department.js

```
const mongoose = require('mongoose');

const departmentSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please provide department name'],
    unique: true,
    trim: true,
  },
  code: {
    type: String,
    required: [true, 'Please provide department code'],
    unique: true,
    uppercase: true,
  },
  description: {
    type: String,
    trim: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
});

module.exports = mongoose.model('Department', departmentSchema);
```

7. models/Club.js

```
const mongoose = require('mongoose');

const clubSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please provide club name'],
    unique: true,
    trim: true,
  },
  category: {
    type: String,
    enum: ['technical', 'cultural', 'sports', 'social', 'other'],
    default: 'other',
  },
  description: {
    type: String,
    trim: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
});

module.exports = mongoose.model('Club', clubSchema);
```

8. middleware/auth.js

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');

// Protect routes - verify JWT token
exports.protect = async (req, res, next) =&gt; {
  let token;

  // Check for token in headers
  if (
    req.headers.authorization &amp;&amp;
    req.headers.authorization.startsWith('Bearer')
  ) {
    token = req.headers.authorization.split(' ')[1];
  }

  // Make sure token exists
  if (!token) {
    return res.status(401).json({
      success: false,
      message: 'Not authorized to access this route',
    });
  }

  try {
    // Verify token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Get user from token
    req.user = await User.findById(decoded.id).select('-password');

    if (!req.user) {
      return res.status(401).json({
        success: false,
        message: 'User not found',
      });
    }

    next();
  } catch (error) {
    return res.status(401).json({
      success: false,
      message: 'Not authorized to access this route',
    });
  }
};

// Grant access to specific roles
exports.authorize = (...roles) =&gt; {
  return (req, res, next) =&gt; {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: `User role ${req.user.role} is not authorized to access this route`,
      });
    }
    next();
  };
};

// Check if user can post notices
exports.canPost = async (req, res, next) =&gt; {
  if (!req.user.canPost &amp;&amp; req.user.role !== 'admin') {
    return res.status(403).json({
      success: false,
      message: 'You do not have permission to post notices',
    });
  }
  next();
};
```

9. middleware/upload.js

```
const multer = require('multer');
const path = require('path');

// Configure storage
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads/');
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, 'notice-' + uniqueSuffix + path.extname(file.originalname));
  }
});

// File filter - only accept PDFs
const fileFilter = (req, file, cb) =&gt; {
  if (file.mimetype === 'application/pdf') {
    cb(null, true);
  } else {
    cb(new Error('Only PDF files are allowed'), false);
  }
};

// Create multer upload instance
const upload = multer({
  storage: storage,
  limits: {
    fileSize: 5 * 1024 * 1024 // 5MB limit
  },
  fileFilter: fileFilter
});

module.exports = upload;
```

10. routes/auth.js

```
const express = require('express');
const router = express.Router();
const jwt = require('jsonwebtoken');
const { body, validationResult } = require('express-validator');
const User = require('../models/User');
const { protect } = require('../middleware/auth');

// Generate JWT Token
const generateToken = (id) =&gt; {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRE,
  });
};

// @route   POST /api/auth/register
// @desc    Register a new user
// @access  Public
router.post(
  '/register',
  [
    body('name').notEmpty().withMessage('Name is required'),
    body('email').isEmail().withMessage('Please provide a valid email'),
    body('password')
      .isLength({ min: 6 })
      .withMessage('Password must be at least 6 characters'),
    body('role')
      .isIn(['student', 'faculty', 'club_member'])
      .withMessage('Invalid role'),
  ],
  async (req, res) =&gt; {
    // Validate input
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ success: false, errors: errors.array() });
    }

    try {
      const { name, email, password, role, department, club } = req.body;

      // Check if user already exists
      let user = await User.findOne({ email });
      if (user) {
        return res.status(400).json({
          success: false,
          message: 'User already exists with this email',
        });
      }

      // Create user
      user = await User.create({
        name,
        email,
        password,
        role,
        department,
        club,
      });

      // Generate token
      const token = generateToken(user._id);

      res.status(201).json({
        success: true,
        token,
        user: {
          id: user._id,
          name: user.name,
          email: user.email,
          role: user.role,
          canPost: user.canPost,
        },
      });
    } catch (error) {
      res.status(500).json({
        success: false,
        message: error.message,
      });
    }
  }
);

// @route   POST /api/auth/login
// @desc    Login user
// @access  Public
router.post(
  '/login',
  [
    body('email').isEmail().withMessage('Please provide a valid email'),
    body('password').notEmpty().withMessage('Password is required'),
  ],
  async (req, res) =&gt; {
    // Validate input
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ success: false, errors: errors.array() });
    }

    try {
      const { email, password } = req.body;

      // Check if user exists
      const user = await User.findOne({ email })
        .select('+password')
        .populate('department')
        .populate('club');

      if (!user) {
        return res.status(401).json({
          success: false,
          message: 'Invalid credentials',
        });
      }
      // Check password
      const isMatch = await user.matchPassword(password);
      if (!isMatch) {
        return res.status(401).json({
          success: false,
          message: 'Invalid credentials',
        });
      }

      // Generate token
      const token = generateToken(user._id);

      res.json({
        success: true,
        token,
        user: {
          id: user._id,
          name: user.name,
          email: user.email,
          role: user.role,
          department: user.department,
          club: user.club,
          canPost: user.canPost,
        },
      });
    } catch (error) {
      res.status(500).json({
        success: false,
        message: error.message,
      });
    }
  }
);

// @route   GET /api/auth/me
// @desc    Get current user
// @access  Private
router.get('/me', protect, async (req, res) =&gt; {
  try {
    const user = await User.findById(req.user.id)
      .populate('department')
      .populate('club');

    res.json({
      success: true,
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
        role: user.role,
        department: user.department,
        club: user.club,
        canPost: user.canPost,
      },
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

module.exports = router;
```

11. routes/users.js

```
const express = require('express');
const router = express.Router();
const User = require('../models/User');
const { protect, authorize } = require('../middleware/auth');

// @route   GET /api/users
// @desc    Get all users (admin only)
// @access  Private/Admin
router.get('/', protect, authorize('admin'), async (req, res) =&gt; {
  try {
    const users = await User.find()
      .populate('department')
      .populate('club')
      .select('-password');

    res.json({
      success: true,
      count: users.length,
      users,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   PUT /api/users/:id/privileges
// @desc    Update user posting privileges (admin only)
// @access  Private/Admin
router.put('/:id/privileges', protect, authorize('admin'), async (req, res) =&gt; {
  try {
    const { canPost } = req.body;

    const user = await User.findByIdAndUpdate(
      req.params.id,
      { canPost },
      { new: true, runValidators: true }
    ).select('-password');

    if (!user) {
      return res.status(404).json({
        success: false,
        message: 'User not found',
      });
    }

    res.json({
      success: true,
      user,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   PUT /api/users/:id/role
// @desc    Update user role (admin only)
// @access  Private/Admin
router.put('/:id/role', protect, authorize('admin'), async (req, res) =&gt; {
  try {
    const { role } = req.body;

    if (!['student', 'faculty', 'club_member', 'admin'].includes(role)) {
      return res.status(400).json({
        success: false,
        message: 'Invalid role',
      });
    }

    const user = await User.findByIdAndUpdate(
      req.params.id,
      { role },
      { new: true, runValidators: true }
    ).select('-password');

    if (!user) {
      return res.status(404).json({
        success: false,
        message: 'User not found',
      });
    }

    res.json({
      success: true,
      user,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

module.exports = router;
```

12. routes/notices.js

```
const express = require('express');
const router = express.Router();
const Notice = require('../models/Notice');
const { protect, canPost } = require('../middleware/auth');
const upload = require('../middleware/upload');

// @route   GET /api/notices
// @desc    Get all notices (with filters)
// @access  Public
router.get('/', async (req, res) =&gt; {
  try {
    const { type, department, club } = req.query;

    // Build query
    let query = { isActive: true };

    if (type) {
      query.type = type;
    }
    if (department) {
      query.department = department;
    }
    if (club) {
      query.club = club;
    }

    const notices = await Notice.find(query)
      .populate('postedBy', 'name role')
      .populate('department', 'name code')
      .populate('club', 'name category')
      .sort('-createdAt');

    res.json({
      success: true,
      count: notices.length,
      notices,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   GET /api/notices/:id
// @desc    Get single notice
// @access  Public
router.get('/:id', async (req, res) =&gt; {
  try {
    const notice = await Notice.findById(req.params.id)
      .populate('postedBy', 'name role')
      .populate('department', 'name code')
      .populate('club', 'name category');

    if (!notice) {
      return res.status(404).json({
        success: false,
        message: 'Notice not found',
      });
    }

    res.json({
      success: true,
      notice,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   POST /api/notices
// @desc    Create a new notice
// @access  Private (Faculty/Club Members with canPost permission)
router.post('/', protect, canPost, upload.single('pdfFile'), async (req, res) =&gt; {
  try {
    const { title, content, type, format, department, club } = req.body;

    // Validate required fields
    if (!title || !type || !format) {
      return res.status(400).json({
        success: false,
        message: 'Please provide title, type, and format',
      });
    }

    // Build notice object
    const noticeData = {
      title,
      type,
      format,
      postedBy: req.user.id,
    };

    // Add content for text notices
    if (format === 'text') {
      if (!content) {
        return res.status(400).json({
          success: false,
          message: 'Content is required for text notices',
        });
      }
      noticeData.content = content;
    }
    // Add PDF file for PDF notices
    if (format === 'pdf') {
      if (!req.file) {
        return res.status(400).json({
          success: false,
          message: 'PDF file is required for PDF notices',
        });
      }
      noticeData.pdfFile = {
        filename: req.file.filename,
        path: req.file.path,
        size: req.file.size,
      };
    }

    // Add department or club based on type
    if (type === 'academic' &amp;&amp; department) {
      noticeData.department = department;
    } else if (type === 'club' &amp;&amp; club) {
      noticeData.club = club;
    }

    const notice = await Notice.create(noticeData);

    const populatedNotice = await Notice.findById(notice._id)
      .populate('postedBy', 'name role')
      .populate('department', 'name code')
      .populate('club', 'name category');

    res.status(201).json({
      success: true,
      notice: populatedNotice,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   PUT /api/notices/:id
// @desc    Update a notice
// @access  Private (Notice owner or admin)
router.put('/:id', protect, async (req, res) =&gt; {
  try {
    let notice = await Notice.findById(req.params.id);

    if (!notice) {
      return res.status(404).json({
        success: false,
        message: 'Notice not found',
      });
    }

    // Check ownership or admin role
    if (
      notice.postedBy.toString() !== req.user.id &amp;&amp;
      req.user.role !== 'admin'
    ) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to update this notice',
      });
    }

    const { title, content, isActive } = req.body;

    notice = await Notice.findByIdAndUpdate(
      req.params.id,
      { title, content, isActive },
      { new: true, runValidators: true }
    )
      .populate('postedBy', 'name role')
      .populate('department', 'name code')
      .populate('club', 'name category');

    res.json({
      success: true,
      notice,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   DELETE /api/notices/:id
// @desc    Delete a notice
// @access  Private (Notice owner or admin)
router.delete('/:id', protect, async (req, res) =&gt; {
  try {
    const notice = await Notice.findById(req.params.id);

    if (!notice) {
      return res.status(404).json({
        success: false,
        message: 'Notice not found',
      });
    }

    // Check ownership or admin role
    if (
      notice.postedBy.toString() !== req.user.id &amp;&amp;
      req.user.role !== 'admin'
    ) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to delete this notice',
      });
    }

    await notice.deleteOne();

    res.json({
      success: true,
      message: 'Notice deleted successfully',
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

module.exports = router;
```

13. routes/departments.js

```
const express = require('express');
const router = express.Router();
const Department = require('../models/Department');
const { protect, authorize } = require('../middleware/auth');

// @route   GET /api/departments
// @desc    Get all departments
// @access  Public
router.get('/', async (req, res) =&gt; {
  try {
    const departments = await Department.find().sort('name');

    res.json({
      success: true,
      count: departments.length,
      departments,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   POST /api/departments
// @desc    Create a new department
// @access  Private/Admin
router.post('/', protect, authorize('admin'), async (req, res) =&gt; {
  try {
    const { name, code, description } = req.body;

    const department = await Department.create({ name, code, description });

    res.status(201).json({
      success: true,
      department,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

module.exports = router;
```

14. routes/clubs.js

```
const express = require('express');
const router = express.Router();
const Club = require('../models/Club');
const { protect, authorize } = require('../middleware/auth');

// @route   GET /api/clubs
// @desc    Get all clubs
// @access  Public
router.get('/', async (req, res) =&gt; {
  try {
    const clubs = await Club.find().sort('name');

    res.json({
      success: true,
      count: clubs.length,
      clubs,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

// @route   POST /api/clubs
// @desc    Create a new club
// @access  Private/Admin
router.post('/', protect, authorize('admin'), async (req, res) =&gt; {
  try {
    const { name, category, description } = req.body;

    const club = await Club.create({ name, category, description });

    res.status(201).json({
      success: true,
      club,
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
});

module.exports = router;
```

Installation Instructions

1. Install Dependencies

```
cd backend
npm install
```

2. Create uploads directory

```
mkdir uploads
```

3. Configure Environment Variables

Create a `.env` file in the backend root directory and update with your values.

4. Start MongoDB

Make sure MongoDB is running on your system or update the `MONGODB_URI` in `.env` to point to your MongoDB instance.

5. Run the Server

```
# Development mode with nodemon
npm run dev

# Production mode
npm start
```

The backend server will run on `http://localhost:5000`

API Endpoints Summary

Authentication

- POST `/api/auth/register` - Register new user
- POST `/api/auth/login` - Login user
- GET `/api/auth/me` - Get current user

Users (Admin only)

- GET `/api/users` - Get all users
- PUT `/api/users/:id/privileges` - Update posting privileges
- PUT `/api/users/:id/role` - Update user role

Notices

- GET `/api/notices` - Get all notices (with filters)
- GET `/api/notices/:id` - Get single notice
- POST `/api/notices` - Create notice (requires canPost)
- PUT `/api/notices/:id` - Update notice
- DELETE `/api/notices/:id` - Delete notice

Departments

- GET `/api/departments` - Get all departments
- POST `/api/departments` - Create department (Admin)

Clubs

- GET `/api/clubs` - Get all clubs
- POST `/api/clubs` - Create club (Admin)