Blogging Platform — Complete Project Code (Minimal, runnable)

```
blogging-platform/
├── backend/
│   ├── package.json
│   ├── .env.example
│   ├── index.js
│   ├── models.js
│   ├── routes/
│   │   ├── auth.js
│   │   ├── posts.js
│   │   └── comments.js
│   ├── middleware/auth.js
│   └── seed.js
└── frontend/
    ├── package.json
    ├── public/index.html
    └── src/
        ├── index.js
        ├── App.js
        ├── api.js
        ├── components/
        │   ├── Header.js
        │   ├── PostList.js
        │   ├── PostView.js
        │   └── NewPost.js
        └── styles.css
```

Backend files

backend/package.json

```
{
  "name": "blogging-backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "seed": "node seed.js"
  },
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.0",
    "sequelize": "^6.32.1",
    "sqlite3": "^5.1.6"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

---

backend/.env.example

```
JWT_SECRET=replace_with_a_secure_secret
PORT=5000
```

---

backend/models.js

```
const { Sequelize, DataTypes } = require('sequelize');
const path = require('path');
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: path.join(__dirname, 'database.sqlite'),
```

```
  logging: false,
});

const User = sequelize.define('User', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  name: { type: DataTypes.STRING, allowNull: false },
  email: { type: DataTypes.STRING, allowNull: false, unique: true },
  passwordHash: { type: DataTypes.STRING, allowNull: false },
  role: { type: DataTypes.STRING, defaultValue: 'user' },
});

const Post = sequelize.define('Post', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  title: { type: DataTypes.STRING, allowNull: false },
  body: { type: DataTypes.TEXT, allowNull: false },
  tags: { type: DataTypes.STRING },
});

const Comment = sequelize.define('Comment', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  body: { type: DataTypes.TEXT, allowNull: false },
  approved: { type: DataTypes.BOOLEAN, defaultValue: true },
});

User.hasMany(Post, { as: 'posts', foreignKey: 'authorId' });
Post.belongsTo(User, { as: 'author', foreignKey: 'authorId' });

Post.hasMany(Comment, { as: 'comments', foreignKey: 'postId' });
Comment.belongsTo(Post, { as: 'post', foreignKey: 'postId' });
Comment.belongsTo(User, { as: 'author', foreignKey: 'authorId' });

module.exports = { sequelize, User, Post, Comment };
```

---

backend/middleware/auth.js

```
const jwt = require('jsonwebtoken');
const { User } = require('../models');
const JWT_SECRET = process.env.JWT_SECRET || 'secret_in_dev';

async function authMiddleware(req, res, next) {
  const auth = req.headers.authorization;
  if (!auth) return res.status(401).json({ error: 'No authorization header' });
  const parts = auth.split(' ');
  if (parts.length !== 2) return res.status(401).json({ error: 'Invalid auth
header' });
```

```
  const token = parts[1];
  try {
    const payload = jwt.verify(token, JWT_SECRET);
    const user = await User.findByPk(payload.id);
    if (!user) return res.status(401).json({ error: 'User not found' });
    req.user = user;
    next();
  } catch (err) {
    return res.status(401).json({ error: 'Invalid token' });
  }
}

module.exports = { authMiddleware };
```

---

backend/routes/auth.js

```
const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { User } = require('../models');
const router = express.Router();
const JWT_SECRET = process.env.JWT_SECRET || 'secret_in_dev';

router.post('/register', async (req, res) => {
  const { name, email, password } = req.body;
  if (!name || !email || !password) return res.status(400).json({ error:
'Missing fields' });
  try {
    const hash = await bcrypt.hash(password, 10);
    const user = await User.create({ name, email, passwordHash: hash });
    return res.json({ id: user.id, name: user.name, email: user.email });
  } catch (err) {
    console.error(err);
    return res.status(400).json({ error: 'Email already in use' });
  }
});

router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) return res.status(400).json({ error: 'Missing
fields' });
  const user = await User.findOne({ where: { email } });
  if (!user) return res.status(401).json({ error: 'Invalid credentials' });
  const ok = await bcrypt.compare(password, user.passwordHash);
  if (!ok) return res.status(401).json({ error: 'Invalid credentials' });
```

```
  const token = jwt.sign({ id: user.id, email: user.email }, JWT_SECRET, {
expiresIn: '7d' });
  return res.json({ token, user: { id: user.id, name: user.name, email:
user.email } });
});

module.exports = router;
```

---

backend/routes/posts.js

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

// List posts (public)
router.get('/', async (req, res) => {
  const posts = await Post.findAll({ include: [{ model: User, as: 'author',
attributes: ['id','name']}], order:[['createdAt','DESC']] });
  res.json(posts);
});

// Get single post with comments
router.get('/:id', async (req, res) => {
  const post = await Post.findByPk(req.params.id, { include: [
    { model: User, as: 'author', attributes: ['id','name']},
    { model: Comment, as: 'comments', include: [{ model: User, as: 'author',
attributes: ['id','name'] } ] }
  ] });
  if (!post) return res.status(404).json({ error: 'Post not found' });
  res.json(post);
});

// Create post (auth)
router.post('/', authMiddleware, async (req, res) => {
  const { title, body, tags } = req.body;
  if (!title || !body) return res.status(400).json({ error: 'Missing title or
body' });
  const post = await Post.create({ title, body, tags, authorId: req.user.id });
  res.json(post);
});

// Edit post (auth, owner)
router.put('/:id', authMiddleware, async (req, res) => {
  const post = await Post.findByPk(req.params.id);
```

```
  if (!post) return res.status(404).json({ error: 'Post not found' });
  if (post.authorId !== req.user.id && req.user.role !== 'admin') return
res.status(403).json({ error: 'Not allowed' });
  const { title, body, tags } = req.body;
  await post.update({ title: title||post.title, body: body||post.body, tags:
tags||post.tags });
  res.json(post);
});

// Delete post
router.delete('/:id', authMiddleware, async (req, res) => {
  const post = await Post.findByPk(req.params.id);
  if (!post) return res.status(404).json({ error: 'Post not found' });
  if (post.authorId !== req.user.id && req.user.role !== 'admin') return
res.status(403).json({ error: 'Not allowed' });
  await post.destroy();
  res.json({ success: true });
});

module.exports = router;
```

---

backend/routes/comments.js

```
const express = require('express');
const { Comment, Post } = require('../models');
const { authMiddleware } = require('../middleware/auth');
const router = express.Router();

// Add comment to post (auth)
router.post('/:postId', authMiddleware, async (req, res) => {
  const post = await Post.findByPk(req.params.postId);
  if (!post) return res.status(404).json({ error: 'Post not found' });
  const { body } = req.body;
  if (!body) return res.status(400).json({ error: 'Comment body required' });
  const comment = await Comment.create({ body, postId: post.id, authorId:
req.user.id, approved: true });
  res.json(comment);
});

module.exports = router;
```

---

backend/seed.js

```
require('dotenv').config();
const bcrypt = require('bcrypt');
const { sequelize, User, Post, Comment } = require('./models');

async function seed(){
  await sequelize.sync({ force: true });
  const hash = await bcrypt.hash('AdminPass123', 10);
  const admin = await User.create({ name: 'Admin', email: 'admin@example.com',
passwordHash: hash, role: 'admin' });
  const uh = await bcrypt.hash('Password123', 10);
  const user = await User.create({ name: 'Demo User', email:
'user@example.com', passwordHash: uh });

  const p1 = await Post.create({ title: 'Welcome to the Blog', body: 'This is
the first post.', tags: 'welcome,intro', authorId: admin.id });
  const p2 = await Post.create({ title: 'Second Post', body: 'Another post for
testing.', tags: 'test', authorId: user.id });

  await Comment.create({ body: 'Nice post!', postId: p1.id, authorId:
user.id });
  console.log('Seed complete');
  process.exit(0);
}

seed();
```

---

backend/index.js

```
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const { sequelize } = require('./models');
const authRoutes = require('./routes/auth');
const postsRoutes = require('./routes/posts');
const commentsRoutes = require('./routes/comments');

const app = express();
app.use(bodyParser.json());
app.use(cors());

app.use('/api/auth', authRoutes);
```

```
app.use('/api/posts', postsRoutes);
app.use('/api/comments', commentsRoutes);

app.get('/health', (req,res)=> res.json({ ok: true }));

const PORT = process.env.PORT || 5000;

sequelize.sync().then(()=>{
  app.listen(PORT, ()=> console.log('Server listening on', PORT));
});
```

Frontend files

frontend/package.json

```
{
  "name": "blogging-frontend",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-scripts": "5.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom"
  }
}
```

---

frontend/public/index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Blogging Platform</title>
  </head>
  <body>
    <div id="root"></div>
```

```
  </body>
</html>
```

---

frontend/src/index.js

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import App from './App';
import './styles.css';

const root = createRoot(document.getElementById('root'));
root.render(<App />);
```

---

frontend/src/api.js

```
const API = (process.env.REACT_APP_API_URL) ? process.env.REACT_APP_API_URL :
'http://localhost:5000';

async function request(path, { method='GET', body, token } = {}){
  const headers = { 'Content-Type': 'application/json' };
  if (token) headers['Authorization'] = 'Bearer ' + token;
  const res = await fetch(API + path, { method, headers, body: body?
JSON.stringify(body): undefined });
  const data = await res.json();
  if (!res.ok) throw data;
  return data;
}

export async function register(name,email,password){
  return request('/api/auth/register', { method: 'POST', body: {
name,email,password } });
}
export async function login(email,password){
  return request('/api/auth/login', { method: 'POST', body: {
email,password } });
}
export async function fetchPosts(){
  return request('/api/posts');
}
export async function fetchPost(id){
  return request('/api/posts/' + id);
}
```

```
export async function createPost({ title, body, tags }, token){
  return request('/api/posts', { method: 'POST', body: { title, body, tags },
token });
}
export async function addComment(postId, body, token){
  return request('/api/comments/' + postId, { method: 'POST', body: { body },
token });
}
```

---

frontend/src/styles.css

```
body{ font-family: Arial, Helvetica, sans-serif; margin:0; padding:0; }
.container{ max-width:900px; margin:20px auto; padding:0 16px; }
.header{ display:flex; justify-content:space-between; align-items:center;
padding:12px 0; }
.header a{ margin-left:12px; }
.post{ border:1px solid #ddd; padding:12px; margin-bottom:12px; border-radius:
6px; }
form input, form textarea{ width:100%; padding:8px; margin-bottom:8px; }
button{ padding:8px 12px; cursor:pointer; }
```

---

frontend/src/components/Header.js

```
import React from 'react';
export default function Header({ user, onLogout, onShowNew }){
  return (
    <div className="header container">
      <div>
        <strong>Blogging Platform</strong>
      </div>
      <div>
        {user ? (
          <>
            <span>Hi, {user.name}</span>
            <button onClick={onShowNew} style={{marginLeft:8}}>New Post</button>
            <button onClick={onLogout} style={{marginLeft:8}}>Logout</button>
          </>
        ) : (
          <>
            <a href="#login">Login</a>
          </>
        )}
```

```
      </div>
    </div>
  );
}
```

---

frontend/src/components/PostList.js

```
import React from 'react';
export default function PostList({ posts, onOpen }){
  return (
    <div className="container">
      {posts.map(p=> (
        <div key={p.id} className="post">
          <h3 style={{margin:0}}>{p.title}</h3>
          <small>by {p.author? p.author.name : 'Unknown'}</small>
          <p>{p.body.substring(0,200)}{p.body.length>200?'...':''}</p>
          <button onClick={()=>onOpen(p.id)}>Read</button>
        </div>
      ))}
    </div>
  );
}
```

---

frontend/src/components/PostView.js

```
import React, { useState, useEffect } from 'react';
import { fetchPost, addComment } from '../api';

export default function PostView({ id, token, user, onBack }){
  const [post, setPost] = useState(null);
  const [comment, setComment] = useState('');
  useEffect(()=>{
    fetchPost(id).then(setPost).catch(err=>console.error(err));
  },[id]);

  if (!post) return <div className="container">Loading...</div>;

  async function submitComment(){
    try{
      await addComment(id, comment, token);
      const updated = await fetchPost(id);
      setPost(updated);
```

```
      setComment('');
    }catch(err){
      alert(err.error || 'Failed');
    }
  }

  return (
    <div className="container">
      <button onClick={onBack}>Back</button>
      <h2>{post.title}</h2>
      <small>by {post.author?post.author.name:'Unknown'}</small>
      <div style={{marginTop:12}} dangerouslySetInnerHTML={{__html:
post.body.replace(/\n/g,'<br/>')}} />

      <h4>Comments</h4>
      {post.comments && post.comments.map(c=> (
        <div key={c.id} style={{borderTop:'1px solid #eee', padding:'8px 0'}}>
          <small>{c.author?c.author.name:'Anon'}</small>
          <p style={{margin:4}}>{c.body}</p>
        </div>
      ))}

      {token ? (
        <div>
          <textarea value={comment} onChange={e=>setComment(e.target.value)}
placeholder="Write a comment" />
          <button onClick={submitComment}>Submit</button>
        </div>
      ) : (
        <p>Login to comment.</p>
      )}
    </div>
  );
}
```

---

frontend/src/components/NewPost.js

```
import React, { useState } from 'react';
import { createPost } from '../api';

export default function NewPost({ token, onCreated, onCancel }){
  const [title,setTitle]=useState('');
  const [body,setBody]=useState('');
  const [tags,setTags]=useState('');
```

```
  async function submit(){
    try{
      await createPost({ title, body, tags }, token);
      onCreated();
    }catch(err){
      alert(err.error || 'Failed');
    }
  }

  return (
    <div className="container">
      <h3>New Post</h3>
      <input placeholder="Title" value={title}
onChange={e=>setTitle(e.target.value)} />
      <textarea placeholder="Body" value={body}
onChange={e=>setBody(e.target.value)} rows={8} />
      <input placeholder="tags (comma separated)" value={tags}
onChange={e=>setTags(e.target.value)} />
      <button onClick={submit}>Publish</button>
      <button onClick={onCancel} style={{marginLeft:8}}>Cancel</button>
    </div>
  );
}
```

---

frontend/src/App.js

```
import React, { useState, useEffect } from 'react';
import Header from './components/Header';
import PostList from './components/PostList';
import PostView from './components/PostView';
import NewPost from './components/NewPost';
import * as api from './api';

function App(){
  const [posts, setPosts] = useState([]);
  const [viewId, setViewId] = useState(null);
  const [showNew, setShowNew] = useState(false);
  const [token, setToken] = useState(localStorage.getItem('token'));
  const [user, setUser] = useState(JSON.parse(localStorage.getItem('user') ||
'null'));

  async function load(){
    try{
      const data = await api.fetchPosts();
      setPosts(data);
```

```
    }catch(err){ console.error(err); }
  }
  useEffect(()=>{ load(); }, []);

  async function handleLogin(){
    const email = prompt('Email (admin@example.com or user@example.com)');
    const pass = prompt('Password (AdminPass123 or Password123)');
    try{
      const res = await api.login(email, pass);
      setToken(res.token);
      setUser(res.user);
      localStorage.setItem('token', res.token);
      localStorage.setItem('user', JSON.stringify(res.user));
      load();
    }catch(err){ alert(err.error || 'Login failed'); }
  }

  function handleLogout(){
    setToken(null); setUser(null); localStorage.removeItem('token');
localStorage.removeItem('user');
  }

  return (
    <div>
      <Header user={user} onLogout={handleLogout} onShowNew={()=>
setShowNew(true)} />

      {!token && <div className="container"><button onClick={handleLogin}
>Login</button></div>}

      {showNew ? (
        <NewPost token={token} onCreated={()=>{ setShowNew(false); load(); }}
onCancel={()=> setShowNew(false)} />
      ) : viewId ? (
        <PostView id={viewId} token={token} user={user} onBack={()=>
setViewId(null)} />
      ) : (
        <PostList posts={posts} onOpen={id=> setViewId(id)} />
      )}
    </div>
  );
}

export default App;
```

Quick setup instructions

1. Create the folders `backend` and `frontend` and paste each file above in the correct path.
2. Backend: in `backend` folder run:
3. `npm install`
4. copy `.env.example` to `.env` and set `JWT_SECRET` (or use default in dev)
5. `npm run seed` to create database and seed users/posts
6. `npm start` to run server on port 5000
7. Frontend: in `frontend` folder run:
8. `npm install`
9. `npm start` (creates React dev server on port 3000)
10. Open `http://localhost:3000` and test the app. Use seeded accounts:
11. Admin: `admin@example.com` / `AdminPass123`
12. User: `user@example.com` / `Password123`
