Building a Full-Stack Resource Sharing Platform

A Complete Guide for Your Portfolio Project

This guide helps beginners create their own resource sharing platform from scratch. Perfect for a portfolio piece that showcases full-stack development skills.

Project Overview

You'll build a platform where users can:

- Share and request resources
- Register/login with JWT auth
- Upload images
- Get notifications
- Track credits/points

Tech stack you'll learn:

- Frontend: React, Material-UI
- Backend: Node.js, Express
- Database: PostgreSQL
- DevOps: Docker, Docker Compose
- Auth: JWT tokens
- File handling: Image uploads

Part 1: Project Setup & Planning (2-3 hours)

1. Create project structure:

```
mkdir my-resource-platform
cd my-resource-platform
```

2. Initialize git:

```
git init
git branch -M main
```

3. Create basic folders:

```
mkdir server client
```

4. Create `.gitignore`:

```
# Node
node_modules/
.env

# React build
client/build/

# Uploads
server/uploads/

# IDE
.vscode/
```

Part 2: Backend Development (4-6 hours)

Step 1: Express Server Setup

1. Initialize server:

```
cd server
npm init -y
```

2. Install core dependencies:

```
npm install express cors dotenv pg bcrypt jsonwebtoken
npm install --save-dev nodemon
```

3. Create basic `server/index.js`:

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';

dotenv.config();
const app = express();
app.use(cors());
app.use(express.json());

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

4. Add to `package.json`:

```
{
  "type": "module",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  }
}
```

Step 2: Database Setup (1-2 hours)

1. Create `server/init.sql`:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    credits INTEGER DEFAULT 5
);

CREATE TABLE resources (
    id SERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    category VARCHAR(100),
    image_filename VARCHAR(200),
    user_id INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE requests (
    id SERIAL PRIMARY KEY,
    resource_id INTEGER REFERENCES resources(id),
    requester_id INTEGER REFERENCES users(id),
    status VARCHAR(20) DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE notifications (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    message TEXT NOT NULL,
    read BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. Create `server/db.js`:

```
import pg from 'pg';
const { Pool } = pg;

export const pool = new Pool({
    user: process.env.DB_USER || 'postgres',
    password: process.env.DB_PASSWORD,
    host: process.env.DB_HOST || 'localhost',
    port: process.env.DB_PORT || 5432,
    database: process.env.DB_DATABASE || 'resources_db'
});
```

Step 3: Auth Routes (2-3 hours)

1. Create `server/middleware/authMiddleware.js`:

```
import jwt from 'jsonwebtoken';

export function authMiddleware(req, res, next) {
    try {
        const token = req.headers.authorization.split(' ')[1];
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded;
        next();
    } catch (err) {
        res.status(401).json({ message: 'Auth failed' });
    }
}
```

2. Create `server/routes/auth.js`:

```
import express from 'express';
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import { pool } from '../db.js';

const router = express.Router();

router.post('/register', async (req, res) => {
    try {
        const { name, email, password } = req.body;
        const hash = await bcrypt.hash(password, 10);
        const result = await pool.query(
            'INSERT INTO users (name, email, password) VALUES ($1, $2, $3) RETURNING id, name
            [name, email, hash]
        );
        res.status(201).json(result.rows[0]);
    } catch (err) {
        res.status(500).json({ message: 'Registration failed' });
    }
});

router.post('/login', async (req, res) => {
    try {
        const { email, password } = req.body;
        const result = await pool.query('SELECT * FROM users WHERE email = $1', [email]);
        if (!result.rows.length) {
            return res.status(401).json({ message: 'Auth failed' });
        }
        const user = result.rows[0];
        const match = await bcrypt.compare(password, user.password);
        if (!match) {
            return res.status(401).json({ message: 'Auth failed' });
        }
        const token = jwt.sign({ id: user.id, email: user.email }, process.env.JWT_SECRET, {
        res.json({ token });
    } catch (err) {
        res.status(500).json({ message: 'Login failed' });
    }
});

export default router;
```

Part 3: Frontend Development (6-8 hours)

Step 1: React Setup

1. Create React app:

```
npx create-react-app client
cd client
```

2. Install dependencies:

```
npm install @mui/material @emotion/react @emotion/styled axios react-router-dom
```

3. Create API client (`client/src/api.js`):

```
import axios from 'axios';

const API = axios.create({ baseURL: 'http://localhost:5000/api' });

API.interceptors.request.use(config => {
    const token = localStorage.getItem('token');
    if (token) {
        config.headers = config.headers || {};
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
});

export default API;
```

Step 2: Components & Routes

1. Create essential components:

- `components/ItemCard.js` - Display a resource
- `components/NotificationBell.js` - Show notifications
- `pages/Home.js` - List resources
- `pages/Login.js` - Auth form
- `pages/CreateResource.js` - Resource form

2. Update `App.js` with routes:

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Home from './pages/Home';
import Login from './pages/Login';
import CreateResource from './pages/CreateResource';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/create" element={<CreateResource />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Part 4: Docker & Deployment (2-3 hours)

1. Create `server/Dockerfile`:

```
FROM node:18-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 5000
CMD ["npm", "run", "dev"]
```

2. Create `client/Dockerfile`:

```
FROM node:18-alpine AS build
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /usr/src/app/build /usr/share/nginx/html
```

```
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

3. Create `docker-compose.yml`:

```
version: "3.8"
services:
  db:
    image: postgres:14-alpine
    environment:
      POSTGRES_USER: ${DB_USER:-postgres}
      POSTGRES_PASSWORD: ${DB_PASSWORD:-changeme}
      POSTGRES_DB: ${DB_DATABASE:-resources_db}
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./server/init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5432:5432"

  server:
    build: ./server
    depends_on:
      - db
    environment:
      - DB_USER=${DB_USER:-postgres}
      - DB_PASSWORD=${DB_PASSWORD:-changeme}
      - DB_HOST=db
      - DB_DATABASE=${DB_DATABASE:-resources_db}
      - JWT_SECRET=${JWT_SECRET:-your-secret-key}
    volumes:
      - ./server:/usr/src/app
      - /usr/src/app/node_modules
    ports:
      - "5000:5000"

  client:
    build: ./client
    ports:
      - "3000:80"

volumes:
  postgres_data:
```

Learning Objectives & Skills Gained

1. Backend Development:

- REST API design

- JWT authentication

- File uploads

- Database modeling

- Middleware patterns

2. Frontend Development:
- React hooks & context
- Material-UI theming
- Form handling
- File upload UI
- Protected routes

3. DevOps & Infrastructure:
- Docker containerization
- Multi-container orchestration
- Environment variables
- Production builds

4. Database:
- Schema design
- Relationships
- Transactions
- Connection pooling

Portfolio Presentation Tips

1. README highlights:
- Problem solved
- Tech stack & architecture
- Key features
- Setup instructions
- API documentation

2. Screenshots/GIFs of:
- Resource listing
- Upload flow
- Notifications
- Mobile responsiveness

3. Code organization:

- Clean folder structure
- Consistent naming
- Error handling
- Security practices

4. Extensions to showcase:

- Image processing
- Unit tests
- CI/CD pipeline
- Cloud deployment

Complete Code Examples

Backend Implementation Examples

Authentication System

The authentication system uses JWT tokens and includes rate limiting for security:

```
// auth.js
import express from 'express';
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import { pool } from '../db.js';

const router = express.Router();

// Input validation helper
function validateRegistration(body) {
    const { name, email, password } = body;
    if (!name || !email || !password) {
        return { valid: false, message: 'All fields required' };
    }
    if (password.length < 6) {
        return { valid: false, message: 'Password must be 6+ characters' };
    }
    return { valid: true };
}

// Register endpoint with validation
router.post('/register', async (req, res) => {
    try {
        const validation = validateRegistration(req.body);
        if (!validation.valid) {
            return res.status(400).json({ message: validation.message });
        }

        const { name, email, password } = req.body;
        const hash = await bcrypt.hash(password, 10);
        const result = await pool.query(
            'INSERT INTO users (name, email, password) VALUES ($1, $2, $3) RETURNING id, nam
            [name, email, hash]
        );

        res.status(201).json(result.rows[0]);
    } catch (err) {
        res.status(500).json({ message: 'Registration failed' });
    }
});

// Login with rate limiting
const loginAttempts = new Map();
router.post('/login', async (req, res) => {
    try {
        const { email, password } = req.body;

        const attempts = loginAttempts.get(email) || 0;
        if (attempts >= 5) {
            return res.status(429).json({ message: 'Too many attempts' });
        }

        const result = await pool.query('SELECT * FROM users WHERE email = $1', [email]);
        if (!result.rows.length) {
            loginAttempts.set(email, attempts + 1);
            return res.status(401).json({ message: 'Auth failed' });
        }

        const user = result.rows[0];
        const match = await bcrypt.compare(password, user.password);
        if (!match) {
            loginAttempts.set(email, attempts + 1);
            return res.status(401).json({ message: 'Auth failed' });
        }

        loginAttempts.delete(email);
        const token = jwt.sign(
            { id: user.id, email: user.email },
            process.env.JWT_SECRET,
            { expiresIn: '24h' }
        );

        res.json({ token, user: {
            id: user.id,
            name: user.name,
            email: user.email,
            credits: user.credits
        }});
    } catch (err) {
        res.status(500).json({ message: 'Login failed' });
    }
});

export default router;
```

Resource Management

Example of resource listing with pagination and filtering:

```
// resources.js
import express from 'express';
import { pool } from '../db.js';
import { authMiddleware } from '../middleware/authMiddleware.js';

const router = express.Router();

router.get('/', async (req, res) => {
    try {
        const { page = 1, limit = 10, category } = req.query;
        const offset = (page - 1) * limit;

        let query = `
            SELECT r.*, u.name as owner_name
            FROM resources r
            JOIN users u ON r.user_id = u.id
        `;
        const params = [];

        if (category) {
            query += ' WHERE r.category = $1';
            params.push(category);
        }

        query += ` ORDER BY r.created_at DESC LIMIT $${params.length + 1} OFFSET $${params.le
        params.push(limit, offset);

        const result = await pool.query(query, params);

        // Add image URLs
        const resources = result.rows.map(r => ({
            ...r,
            image_url: r.image_filename ? `/uploads/${r.image_filename}` : null
        }));

        // Get total count
        const countResult = await pool.query(
            'SELECT COUNT(*) FROM resources' + (category ? ' WHERE category = $1' : ''),
            category ? [category] : []
        );

        res.json({
            resources,
            pagination: {
                page: parseInt(page),
                limit: parseInt(limit),
                total: parseInt(countResult.rows[0].count),
                pages: Math.ceil(countResult.rows[0].count / limit)
            }
        });
    } catch (err) {
        res.status(500).json({ message: 'Failed to fetch resources' });
    }
});

export default router;
```

Frontend Implementation Examples

Authentication Context

React context for managing authentication state:

```
// AuthContext.js
import React, { createContext, useState, useContext, useEffect } from 'react';

const AuthContext = createContext();

export function AuthProvider({ children }) {
    const [user, setUser] = useState(null);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const token = localStorage.getItem('token');
        const userData = localStorage.getItem('user');
        if (token && userData) {
            setUser(JSON.parse(userData));
        }
        setLoading(false);
    }, []);

    const login = async (email, password) => {
        try {
            const res = await fetch('/api/auth/login', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ email, password })
            });

            if (!res.ok) throw new Error('Login failed');

            const data = await res.json();
            localStorage.setItem('token', data.token);
            localStorage.setItem('user', JSON.stringify(data.user));
            setUser(data.user);
            return data.user;
        } catch (err) {
            throw err;
        }
    };

    const logout = () => {
        localStorage.removeItem('token');
        localStorage.removeItem('user');
        setUser(null);
    };

    return (
        <AuthContext.Provider value={{ user, loading, login, logout }}>
            {!loading && children}
        </AuthContext.Provider>
    );
}

export const useAuth = () => useContext(AuthContext);
```

Resource Creation Form

Example of a form with image upload:

```
// CreateResource.js
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

export default function CreateResource() {
    const navigate = useNavigate();
    const { user } = useAuth();
    const [loading, setLoading] = useState(false);
    const [form, setForm] = useState({
        title: '',
        description: '',
        category: 'other',
        image: null
    });

    const handleSubmit = async (e) => {
        e.preventDefault();
        setLoading(true);

        try {
            let imageFilename = null;

            if (form.image) {
                const formData = new FormData();
                formData.append('file', form.image);

                const uploadRes = await fetch('/api/uploads', {
                    method: 'POST',
                    headers: {
                        'Authorization': `Bearer ${localStorage.getItem('token')}`
                    },
                    body: formData
                });

                if (!uploadRes.ok) throw new Error('Upload failed');

                const uploadData = await uploadRes.json();
                imageFilename = uploadData.filename;
            }

            const res = await fetch('/api/resources', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${localStorage.getItem('token')}`
                },
                body: JSON.stringify({
                    title: form.title,
                    description: form.description,
                    category: form.category,
                    image_filename: imageFilename
                })
            });

            if (!res.ok) throw new Error('Failed to create resource');

            navigate('/dashboard');
        } catch (err) {
            console.error('Create resource error:', err);
        } finally {
            setLoading(false);
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            {/* Form implementation */}
        </form>
    );
}
```

Docker Configuration Examples

Production Docker Setup

Multi-container setup with Nginx:

```
# docker-compose.yml
version: '3.8'
services:
  db:
    image: postgres:14-alpine
    environment:
      POSTGRES_DB: resourcedb
      POSTGRES_USER: dbuser
      POSTGRES_PASSWORD: dbpassword
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./server/init.sql:/docker-entrypoint-initdb.d/init.sql
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U dbuser -d resourcedb"]
      interval: 5s
      timeout: 5s
      retries: 5

  server:
    build:
      context: ./server
      dockerfile: Dockerfile
    environment:
      NODE_ENV: production
      DB_HOST: db
      JWT_SECRET: ${JWT_SECRET}
    volumes:
      - ./server/uploads:/app/uploads
    depends_on:
      db:
        condition: service_healthy

  client:
    build:
      context: ./client
      dockerfile: Dockerfile
    ports:
      - "80:80"
    depends_on:
      - server

volumes:
  postgres_data:
```

Backend Dockerfile

Optimized Node.js container:

```
# server/Dockerfile
FROM node:18-alpine

WORKDIR /app
RUN mkdir -p uploads && chmod 755 uploads

COPY package*.json ./
RUN npm ci --only=production
COPY . .
```

```
USER node
CMD ["node", "src/index.js"]
```

API Documentation

Authentication Endpoints

POST /api/auth/register

Register a new user account.

Request:

```
{
  "name": "string",
  "email": "string",
  "password": "string"
}
```

Response (201):

```
{
  "id": "integer",
  "name": "string",
  "email": "string"
}
```

POST /api/auth/login

Authenticate and get token.

Request:

```
{
  "email": "string",
  "password": "string"
}
```

Response (200):

```
{
  "token": "string",
  "user": {
    "id": "integer",
    "name": "string",
    "email": "string",
    "credits": "integer"
  }
}
```

Resource Endpoints

GET /api/resources

List resources with pagination.

Query Parameters:

- page (optional): Page number (default: 1)
- limit (optional): Items per page (default: 10)
- category (optional): Filter by category

Response (200):

```
{
  "resources": [
    {
      "id": "integer",
      "title": "string",
      "description": "string",
      "category": "string",
      "image_url": "string",
      "owner_name": "string"
    }
  ],
  "pagination": {
    "page": "integer",
    "limit": "integer",
    "total": "integer",
    "pages": "integer"
  }
}
```

POST /api/resources

Create a new resource.

Headers:

- Authorization: Bearer {token}

Request:

```
{
  "title": "string",
  "description": "string",
  "category": "string",
  "image_filename": "string (optional)"
}
```

Response (201):

```
{
  "id": "integer",
  "title": "string",
  "description": "string",
  "category": "string",
  "image_url": "string"
}
```

Common Status Codes

- 200: Success
- 201: Created
- 400: Bad Request
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 429: Too Many Requests
- 500: Internal Server Error

Next Steps & Extensions

1. Add features:
- Search/filters
- User profiles
- Admin dashboard
- Chat system

2. Improve UX:
- Loading states

- Error boundaries
- Form validation
- Animations

3. Infrastructure:
- AWS/GCP deployment
- CDN for images
- SSL/HTTPS
- Monitoring

4. Testing:
- Unit tests (Jest)
- Integration tests
- E2E tests (Cypress)
- Load testing

Remember to commit often and document your learning journey. This project touches many aspects of modern web development, making it an excellent portfolio piece that demonstrates full-stack capabilities.

Would you like me to:
1. Generate a PDF of this guide
2. Add more code examples for any section
3. Create a starter template repository with this structure
4. Add detailed API documentation