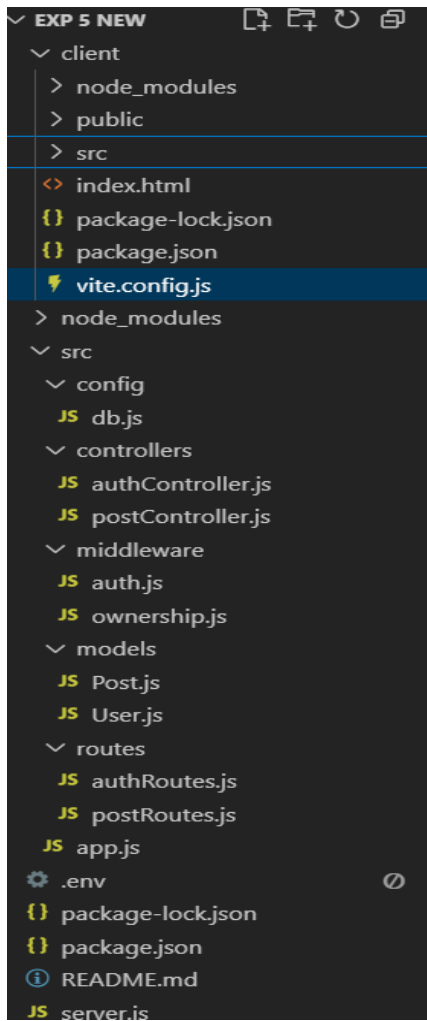


Experiment No. 5: Create secure, production-ready RESTful APIs

Code:-

1. Directory Structure:



2. User Model:

1. User Model (src/models/User.js)

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

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, trim: true, minlength: 3 },
  email: { type: String, required: true, unique: true, lowercase: true },
  password: { type: String, required: true, minlength: 8, select: false },
  role: { type: String, enum: ['user', 'admin'], default: 'user' }
}, { timestamps: true });

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

userSchema.methods.comparePassword = async function (candidatePassword) {
  return bcrypt.compare(candidatePassword, this.password);
};
```

3. JWT Auth Middleware:

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

async function authenticate(req, res, next) {
  try {
    const token = req.headers.authorization?.slice(7);
    if (!token) return res.status(401).json({ message: 'Missing token' });

    const payload = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(payload.sub).select('_id username email role');

    req.user = { id: user._id.toString(), role: user.role };
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid token' });
  }
}
```

3. Auth Controller:

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

async function register(req, res) {
  try {
    const { username, email, password } = req.body;
    const existing = await User.findOne({ email });
    if (existing) return res.status(409).json({ message: 'Email already in use' });

    const user = await User.create({ username, email, password });
    const token = jwt.sign({ sub: user._id }, process.env.JWT_SECRET,
      { expiresIn: process.env.JWT_EXPIRES_IN });

    res.status(201).json({ token, user: { id: user._id, username, email, role: user.role } });
  } catch (err) {
    res.status(500).json({ message: 'Server error' });
  }
}

async function login(req, res) {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email }).select('+password');
    if (!user || !(await user.comparePassword(password))) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const token = jwt.sign({ sub: user._id }, process.env.JWT_SECRET,
      { expiresIn: process.env.JWT_EXPIRES_IN });
    res.json({ token, user: { id: user._id, username: user.username, email, role: user.role } });
  } catch (err) {
    res.status(500).json({ message: 'Server error' });
  }
}
```

4. Post Routes:

```
const express = require('express');
const { authenticate } = require('../middleware/auth');
const { allowAuthorOrAdmin } = require('../middleware/ownership');
const { createPost, getPosts, updatePost, deletePost } = require('../controllers/postController');

const router = express.Router();

router.get('/', getPosts); // Public
router.post('/', authenticate, createPost); // Auth required
router.put('/:id', authenticate, allowAuthorOrAdmin, updatePost);
router.delete('/:id', authenticate, allowAuthorOrAdmin, deletePost);
```

5. API Service:

```
import axios from 'axios';

const api = axios.create({ baseURL: '/' });

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

expo
```

6. Post Component:

```
import React, { useEffect, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import api from '@api';

export default function Posts() {
  const [posts, setPosts] = useState([]);
  const [search, setSearch] = useState('');
  const user = JSON.parse(localStorage.getItem('user') || 'null');

  useEffect(() => {
    async function load() {
      const { data } = await api.get('/posts');
      setPosts(data);
    }
    load();
  }, []);

  const filtered = posts.filter(p =>
    p.title.toLowerCase().includes(search.toLowerCase()) ||
    p.content.toLowerCase().includes(search.toLowerCase())
  );

  return (
    <div className="grid">
      <div className="hero">
        <h1>Blog</h1>
        <input placeholder="Search..." value={search} onChange={(e) => setSearch(e.target.value)} />
        {user && <button onClick={() => navigate('/posts/new')}>New Post</button>}
      </div>

      <div className="grid-articles">
        {filtered.map(post => (
          <article key={post._id} className="card">
            <h3>{post.title}</h3>
            <p>{post.content.slice(0, 160)}...</p>
            <div className="card-footer">
              <span>by {post.author?.username}</span>
              {user?.role === 'admin' || user?.id === post.author?._id ? (
                <div>
                  <Link to={`/${post._id}/edit`}>Edit</Link>
                  <button onClick={() => deletePost(post._id)}>Delete</button>
                </div>
              ) : null}
            </div>
          </article>
        ))}
      </div>
    </div>
  );
}
```

7. Protected Route:

```
function ProtectedRoute({ children }) {  
  const token = localStorage.getItem('token');  
  return token ? children : <Navigate to="/login" replace />;  
}  
  
function App() {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/" element={<Home />} />  
        <Route path="/posts" element={<Posts />} />  
        <Route path="/login" element={<Login />} />  
        <Route path="/register" element={<Register />} />  
        <Route path="/posts/new" element={  
          <ProtectedRoute><PostEdit mode="create" /></ProtectedRoute>  
        } />  
        <Route path="/posts/:id/edit" element={  
          <ProtectedRoute><PostEdit mode="edit" /></ProtectedRoute>  
        } />  
      </Routes>  
    </BrowserRouter>  
  );  
}
```

8. Server Bootstrap:

```
require('dotenv').config();  
const app = require('./src/app');  
const connectDB = require('./src/config/db');  
  
async function start() {  
  await connectDB(process.env.MONGO_URI);  
  app.listen(process.env.PORT || 3000, () => {  
    console.log(`Server running on port ${process.env.PORT || 3000}`);  
  });  
}  
  
start();
```

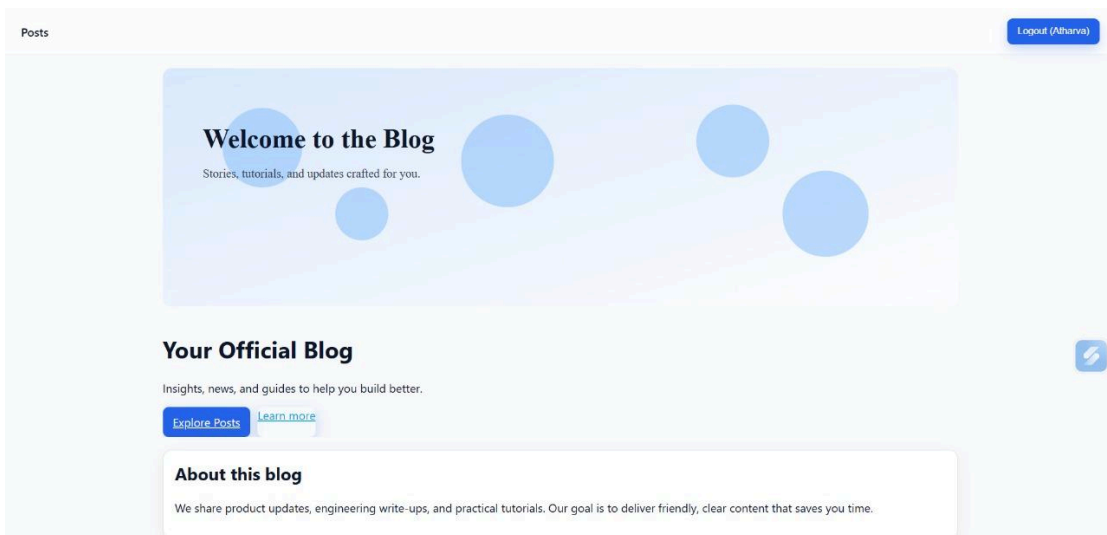
Output:

1.



A screenshot of a web application's registration page. At the top, a navigation bar contains the links "Posts", "Login", and "Register". The main content area is titled "Register" and features a form with three input fields: "Username" (containing "Alharva"), "Email", and "Password" (masked with asterisks). A "Register" button is positioned below the password field.

2.



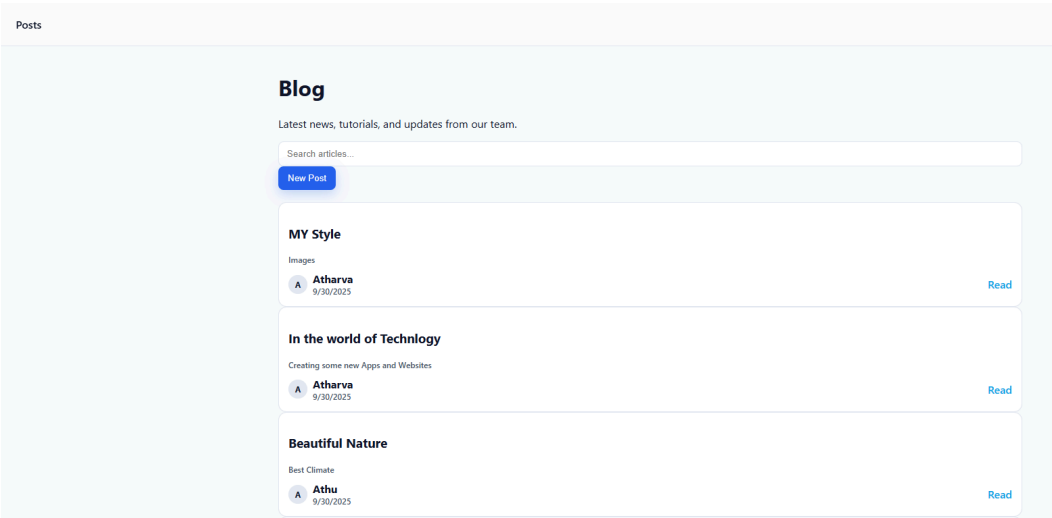
A screenshot of the home page of a web application. The navigation bar includes "Posts" and a "Logout (Alharva)" button. The main content area features a large banner with the text "Welcome to the Blog" and "Stories, tutorials, and updates crafted for you." Below the banner, there is a section titled "Your Official Blog" with the subtitle "Insights, news, and guides to help you build better." This section includes two buttons: "Explore Posts" and "Learn more". At the bottom, there is a box titled "About this blog" with the text "We share product updates, engineering write-ups, and practical tutorials. Our goal is to deliver friendly, clear content that saves you time."

3.

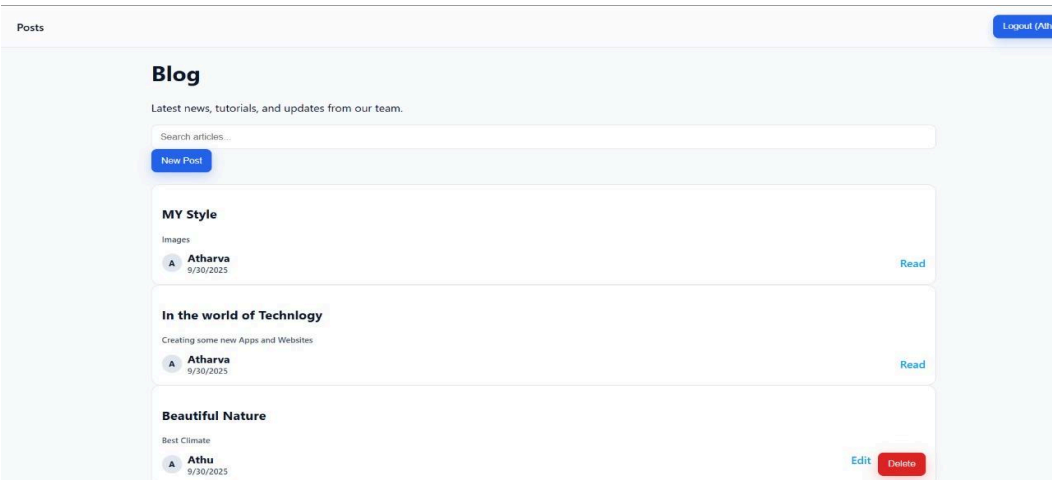


A screenshot of a web application's "New Post" form. The navigation bar shows "Posts" and a "Logout (Alharva)" button. The main content area is titled "New Post" and contains a form with a "Title" field (containing "In Traditional Form") and a "Content" field (containing "Great Outings"). A "Save" button is located at the bottom of the form.

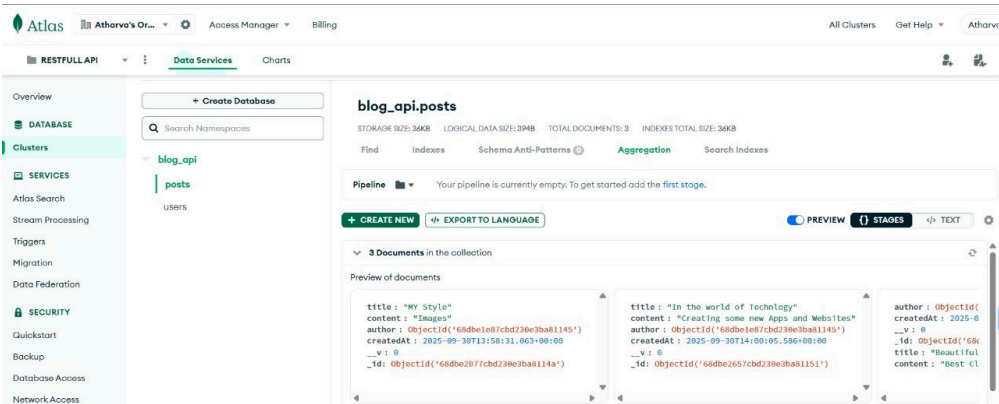
4.



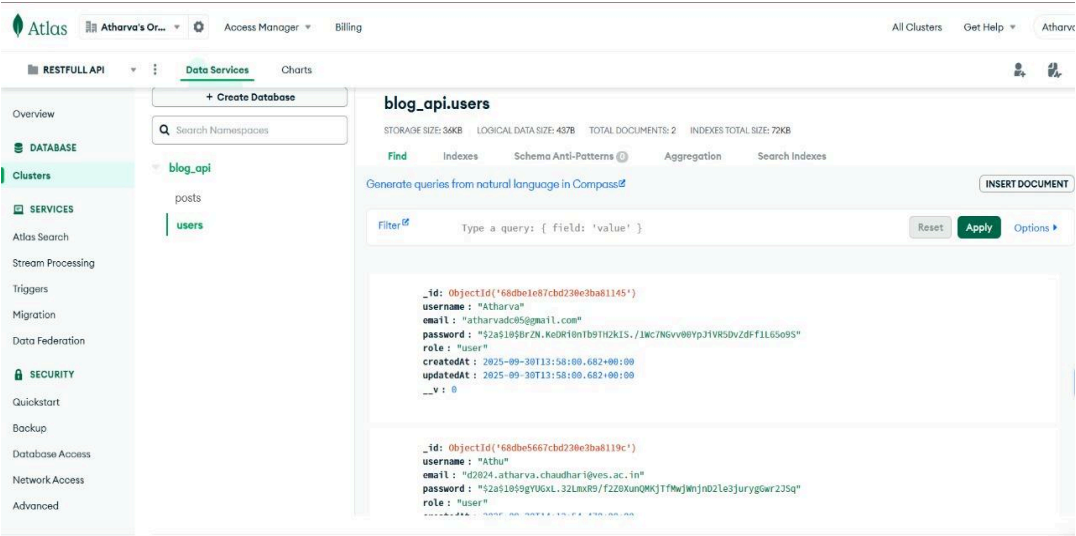
5.



6.



7.



8.

