// backend/middleware/authMiddleware.js

const jwt = require('jsonwebtoken');

const authenticate = (req, res, next) => {

  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {

    return res.status(401).json({ error: 'No or invalid token' });

  }

  const token = authHeader.split(' ')[1];

  try {

    const decoded = jwt.verify(token, process.env.JWT\_SECRET);

    req.user = decoded; // Store user ID in request

    next();

  } catch (err) {

    res.status(401).json({ error: 'Invalid token' });

  }

};

module.exports = authenticate;

authMiddleware.js

// backend/routes/authRoutes.js

const express = require('express');

const pool = require('../utils/db');

const { generateToken, hashPassword, comparePasswords } = require('../utils/auth');

const router = express.Router();

// Register route

router.post('/register', async (req, res) => {

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

  try {

    const hashed = await hashPassword(password);

    const result = await pool.query(

      'INSERT INTO users (name, email, password) VALUES ($1, $2, $3) RETURNING id',

      [name, email, hashed]

    );

    const token = generateToken(result.rows[0].id);

    res.json({ token });

  } catch (err) {

    console.error(err);

    res.status(400).json({ error: 'Registration failed. Email may already be in use.' });

  }

});

// Login route

router.post('/login', async (req, res) => {

  const { email, password } = req.body;

  try {

    const result = await pool.query('SELECT \* FROM users WHERE email = $1', [email]);

    const user = result.rows[0];

    if (!user) return res.status(401).json({ error: 'Invalid credentials' });

    const isMatch = await comparePasswords(password, user.password);

    if (!isMatch) return res.status(401).json({ error: 'Invalid credentials' });

    const token = generateToken(user.id);

    res.json({ token });

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Login failed' });

  }

});

module.exports = router;

authRoutes.js

// backend/routes/scoreRoutes.js

const express = require('express');

const multer = require('multer');

const { parseResume, parseJob, getATSSimilarityScore } = require('../utils/atsHelpers');

const pool = require('../utils/db');

const authenticate = require('../middleware/authMiddleware');

const router = express.Router();

const storage = multer.memoryStorage();

const upload = multer({ storage });

// POST /api/score (protected route)

router.post('/score', authenticate, upload.fields([

  { name: 'resume', maxCount: 1 },

  { name: 'job', maxCount: 1 },

]), async (req, res) => {

  try {

    const resumeFile = req.files['resume'][0];

    const jobFile = req.files['job'][0];

    const resumeText = await parseResume(resumeFile);

    const jobText = await parseJob(jobFile);

    const score = getATSSimilarityScore(resumeText, jobText);

    // Save to DB

    await pool.query(

      'INSERT INTO checks (user\_id, resume\_text, job\_text, score) VALUES ($1, $2, $3, $4)',

      [req.user.id, resumeText, jobText, score]

    );

    res.json({ score });

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Scoring failed' });

  }

});

// GET /api/history (protected route)

router.get('/history', authenticate, async (req, res) => {

  try {

    const result = await pool.query(

      'SELECT id, score, created\_at FROM checks WHERE user\_id = $1 ORDER BY created\_at DESC',

      [req.user.id]

    );

    res.json({ history: result.rows });

  } catch (err) {

    res.status(500).json({ error: 'Could not load history' });

  }

});

module.exports = router;

scoreRoutes.js

const pdfParse = require('pdf-parse');

const mammoth = require('mammoth');

const stopwords = require('./stopwords.json');

function getFileExtension(filename) {

  return filename.split('.').pop().toLowerCase();

}

async function parseResume(file) {

  const ext = getFileExtension(file.originalname);

  if (ext === 'pdf') {

    return (await pdfParse(file.buffer)).text;

  } else if (ext === 'docx') {

    const result = await mammoth.extractRawText({ buffer: file.buffer });

    return result.value;

  } else {

    throw new Error('Unsupported file type');

  }

}

async function parseJob(file) {

  // Reuse the same logic

  return parseResume(file);

}

function cleanText(text) {

  return text

    .toLowerCase()

    .match(/\w+/g)

    .filter(word => !stopwords.includes(word));

}

function getATSSimilarityScore(resumeText, jobText) {

  const resumeWords = new Set(cleanText(resumeText));

  const jobWords = new Set(cleanText(jobText));

  const matched = [...resumeWords].filter(word => jobWords.has(word));

  return Math.round((matched.length / jobWords.size) \* 100);

}

// ✅ Properly export all 3 functions

module.exports = {

  parseResume,

  parseJob,

  getATSSimilarityScore,

};

atsHelpers.js

// backend/utils/auth.js

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

const generateToken = (userId) => {

  return jwt.sign({ id: userId }, process.env.JWT\_SECRET, { expiresIn: '7d' });

};

const hashPassword = async (password) => {

  const salt = await bcrypt.genSalt(10);

  return await bcrypt.hash(password, salt);

};

const comparePasswords = async (rawPassword, hashedPassword) => {

  return await bcrypt.compare(rawPassword, hashedPassword);

};

module.exports = {

  generateToken,

  hashPassword,

  comparePasswords,

};

Auth.js

// backend/utils/db.js

const { Pool } = require('pg');

require('dotenv').config();

const pool = new Pool({

connectionString: process.env.DATABASE\_URL,

ssl: { rejectUnauthorized: false }, // Important for Neon

});

module.exports = pool;

db.js

[

  "the", "is", "in", "at", "of", "a", "and", "to", "an",

  "on", "for", "with", "as", "by", "from", "that", "this",

  "are", "be", "it", "or", "was", "were", "which", "has", "have"

]

Stopwords.json

const express = require('express');

const cors = require('cors');

const dotenv = require('dotenv');

const path = require('path');

const pool = require('./utils/db');

const authRoutes = require('./routes/authRoutes');

const scoreRoutes = require('./routes/scoreRoutes');

dotenv.config();

const app = express();

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

app.use(cors());

app.use(express.json());

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

app.use('/api', scoreRoutes);

// 🔌 Test DB connection route

app.get('/api/db-test', async (req, res) => {

  try {

    const result = await pool.query('SELECT NOW()');

    res.json({ success: true, time: result.rows[0].now });

  } catch (err) {

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

  }

});

app.listen(PORT, () => {

  console.log(`Server running on http://localhost:${PORT}`);

});

Index.js file code

navigation bar ( with login option on the right side , left side we have our website name )

login button

and about the page and research we did on ats scores

when the user clicks on login button he will redirected to login page

here ( currently we have manual login and google account login )

i want to remove the manual login keep only google login

or else we remove the complete login page and we directly use pop up google account login when the user clicks login button ( we need to think about it )

after login

now user will enter into private pages -> user landing page

navigation bar should display user name and profile pic , dashboard ( means user landing page ) and logout

in the user landing page user will able to see his score history and upload button

when he clicks on the upload button he will redirected to upload page now he can upload the resume and job pdf when he click on the submit

user will be able to see the ats score

this is my project code currently im building a ats resume score web site for students to check their resume ats score but while building im facing so many difficulties my main aim is to make this web site as a real world web site i will buy a domain for this web site i will provide you the file code one by one tell me what are the errors or mistakes i have done you need to remember every line of the code i will give you file codes one after another dont expalin anything until i say something like "now its time"  
  
now its time first list all the mistakes i have made and tell me the unused files and codes