**CODE ONLY: ATIFACTS**

// project-code/package.json

{

"name": "task-manager",

"version": "1.0.0",

"scripts": {

"start": "node server/index.js",

"dev": "nodemon server/index.js",

"build": "react-scripts build && electron-builder",

"package:all": "node scripts/package-all.js"

},

"dependencies": {

"express": "^4.17.1",

"react": "^17.0.2",

"electron": "^16.0.0",

"pg": "^8.7.1",

"tensorflow": "^3.0.0",

"docx": "^7.0.0"

}

}

// project-code/client/src/App.js

import React, { useState, useEffect } from 'react';

import { TaskList, TaskForm, Analytics } from './components';

function App() {

const [user, setUser] = useState(null);

const [tasks, setTasks] = useState([]);

const [loading, setLoading] = useState(true);

useEffect(() => {

checkAuth();

initializeWebSocket();

}, []);

return (

<div className="app">

<Navbar user={user} />

<main>

{user ? (

<>

<TaskForm onSubmit={addTask} />

<TaskList tasks={tasks} />

<Analytics data={tasks} />

</>

) : (

<Auth onLogin={handleLogin} />

)}

</main>

</div>

);

}

// project-code/server/index.js

const express = require('express');

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

const WebSocket = require('ws');

const app = express();

const server = http.createServer(app);

const wss = new WebSocket.Server({ server });

app.use(express.json());

app.use(cors());

// Database connection

const pool = new Pool({

connectionString: process.env.DATABASE\_URL,

});

// WebSocket connection

wss.on('connection', (ws) => {

ws.on('message', handleMessage);

ws.on('close', handleClose);

});

// API routes

app.post('/api/tasks', authenticateToken, async (req, res) => {

try {

const { title, description } = req.body;

const result = await pool.query(

'INSERT INTO tasks (title, description, user\_id) VALUES ($1, $2, $3) RETURNING \*',

[title, description, req.user.id]

);

res.json(result.rows[0]);

} catch (error) {

res.status(500).json({ error: error.message });

}

});

// project-code/ai/MLService.js

class MLService {

constructor() {

this.model = new tf.Sequential();

this.setupModel();

}

async analyzeTask(task) {

const embedding = await this.generateEmbedding(task);

const prediction = this.model.predict(embedding);

return this.processResults(prediction);

}

async webSearch(query) {

// Web search implementation

}

}

// project-code/desktop/main.js

const { app, BrowserWindow } = require('electron');

const path = require('path');

const isDev = require('electron-is-dev');

function createWindow() {

const mainWindow = new BrowserWindow({

width: 1200,

height: 800,

webPreferences: {

nodeIntegration: true,

contextIsolation: false

}

});

mainWindow.loadURL(

isDev

? 'http://localhost:3000'

: `file://${path.join(\_\_dirname, '../build/index.html')}`

);

if (isDev) {

mainWindow.webContents.openDevTools();

}

}

app.whenReady().then(createWindow);

// project-code/deployment/docker-compose.yml

version: '3.8'

services:

app:

build: .

ports:

- "3000:3000"

environment:

- NODE\_ENV=production

- DATABASE\_URL=${DATABASE\_URL}

depends\_on:

- db

db:

image: postgres:14

environment:

- POSTGRES\_USER=${DB\_USER}

- POSTGRES\_PASSWORD=${DB\_PASSWORD}

volumes:

- postgres\_data:/var/lib/postgresql/data

// project-code/deployment/k8s/deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: task-manager

spec:

replicas: 3

selector:

matchLabels:

app: task-manager

template:

metadata:

labels:

app: task-manager

spec:

containers:

- name: task-manager

image: task-manager:latest

ports:

- containerPort: 3000

env:

- name: DATABASE\_URL

valueFrom:

secretKeyRef:

name: app-secrets

key: database-url

// project-code/database/schema.sql

CREATE TABLE users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

username VARCHAR(255) NOT NULL UNIQUE,

email VARCHAR(255) NOT NULL UNIQUE,

password\_hash VARCHAR(255) NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE tasks (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

title VARCHAR(255) NOT NULL,

description TEXT,

status VARCHAR(50) DEFAULT 'pending',

user\_id UUID REFERENCES users(id) ON DELETE CASCADE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

// Directory structure and file contents

// 1. Frontend (client/src/)

// App.js

import React, { useState, useEffect } from 'react';

import { Card, CardHeader, CardTitle, CardContent } from '@/components/ui/card';

import { Input } from '@/components/ui/input';

import { Button } from '@/components/ui/button';

import { Alert, AlertDescription } from '@/components/ui/alert';

import {

PlusCircle,

CheckCircle,

Circle,

Trash2,

Calendar,

Tag,

Clock,

Filter,

Search

} from 'lucide-react';

const App = () => {

// State management

const [user, setUser] = useState(null);

const [tasks, setTasks] = useState([]);

const [newTask, setNewTask] = useState({ title: '', dueDate: '', priority: 'medium', tags: [] });

const [loading, setLoading] = useState(false);

const [error, setError] = useState(null);

const [filter, setFilter] = useState('all');

const [searchQuery, setSearchQuery] = useState('');

const [view, setView] = useState('list');

const [loginData, setLoginData] = useState({ username: '', password: '' });

const [isRegistering, setIsRegistering] = useState(false);

// Authentication functions

const handleAuth = async (isLogin) => {

try {

setLoading(true);

const endpoint = isLogin ? '/api/login' : '/api/register';

const response = await fetch(endpoint, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(loginData)

});

if (!response.ok) throw new Error('Authentication failed');

const data = await response.json();

localStorage.setItem('token', data.token);

setUser(data.user);

fetchTasks();

} catch (err) {

handleError(err);

}

};

// Task functions

const addTask = async () => {

try {

setLoading(true);

const response = await fetch('/api/tasks', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${localStorage.getItem('token')}`

},

body: JSON.stringify(newTask)

});

if (!response.ok) throw new Error('Failed to add task');

const task = await response.json();

setTasks([task, ...tasks]);

setNewTask({ title: '', dueDate: '', priority: 'medium', tags: [] });

} catch (err) {

handleError(err);

}

};

// Return JSX

return (

<div className="w-full max-w-4xl mx-auto p-6">

{/\* Component rendering logic \*/}

</div>

);

};

// 2. Backend (server/)

// index.js

const express = require('express');

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

const cors = require('cors');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const WebSocket = require('ws');

const http = require('http');

const app = express();

const server = http.createServer(app);

const wss = new WebSocket.Server({ server });

app.use(cors());

app.use(express.json());

// Database setup

const pool = new Pool({

connectionString: process.env.DATABASE\_URL,

});

// WebSocket handling

wss.on('connection', (ws) => {

ws.on('message', async (message) => {

const data = JSON.parse(message);

// Handle real-time updates

});

});

// API routes

app.post('/api/auth/register', async (req, res) => {

try {

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

const hashedPassword = await bcrypt.hash(password, 10);

const result = await pool.query(

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

[username, email, hashedPassword]

);

const token = jwt.sign({ userId: result.rows[0].id }, process.env.JWT\_SECRET);

res.json({ user: result.rows[0], token });

} catch (error) {

res.status(500).json({ error: error.message });

}

});

// 3. AI Integration (services/ai.js)

class AIService {

constructor() {

this.model = new tf.Sequential();

this.setupModel();

}

async setupModel() {

this.model.add(tf.layers.dense({ units: 64, inputShape: [100] }));

this.model.add(tf.layers.dense({ units: 32, activation: 'relu' }));

this.model.add(tf.layers.dense({ units: 16, activation: 'relu' }));

await this.model.compile({

optimizer: tf.train.adam(),

loss: 'categoricalCrossentropy',

metrics: ['accuracy']

});

}

async analyzeTask(task) {

const embedding = await this.generateEmbedding(task);

return this.model.predict(embedding);

}

}

// 4. Desktop Integration (electron/main.js)

const { app, BrowserWindow } = require('electron');

const path = require('path');

const isDev = require('electron-is-dev');

function createWindow() {

const mainWindow = new BrowserWindow({

width: 1200,

height: 800,

webPreferences: {

nodeIntegration: true,

contextIsolation: false

}

});

mainWindow.loadURL(

isDev

? 'http://localhost:3000'

: `file://${path.join(\_\_dirname, '../build/index.html')}`

);

}

app.whenReady().then(createWindow);

// 5. Database Schema (database/schema.sql)

CREATE TABLE users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

username VARCHAR(255) UNIQUE NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE tasks (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

title VARCHAR(255) NOT NULL,

description TEXT,

status VARCHAR(50) DEFAULT 'pending',

priority VARCHAR(50) DEFAULT 'medium',

due\_date TIMESTAMP WITH TIME ZONE,

user\_id UUID REFERENCES users(id) ON DELETE CASCADE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

tags TEXT[]

);

// 6. Deployment Configuration

// docker-compose.yml

version: '3.8'

services:

app:

build: .

ports:

- "3000:3000"

environment:

- NODE\_ENV=production

- DATABASE\_URL=${DATABASE\_URL}

depends\_on:

- db

db:

image: postgres:14

environment:

- POSTGRES\_USER=${DB\_USER}

- POSTGRES\_PASSWORD=${DB\_PASSWORD}

volumes:

- postgres\_data:/var/lib/postgresql/data

// 7. Export Service

class ExportService {

async exportToWord(data) {

const doc = new Document({

sections: [{

properties: {},

children: [

new Paragraph({

text: "Task Report",

heading: HeadingLevel.HEADING\_1

}),

// Add task data

]

}]

});

const buffer = await Packer.toBuffer(doc);

return buffer;

}

}

// package.json

{

"name": "task-manager",

"version": "1.0.0",

"scripts": {

"start": "node server/index.js",

"dev": "nodemon server/index.js",

"build": "react-scripts build && electron-builder",

"test": "jest"

},

"dependencies": {

"express": "^4.17.1",

"react": "^17.0.2",

"electron": "^16.0.0",

"@tensorflow/tfjs": "^3.0.0",

"docx": "^7.0.0",

"pg": "^8.7.1",

"ws": "^8.2.3"

}

// tests/test-suite.js

const request = require('supertest');

const app = require('../server');

describe('Authentication', () => {

test('should register user', async () => {

const res = await request(app)

.post('/api/auth/register')

.send({

username: 'testuser',

email: 'test@example.com',

password: 'password123'

});

expect(res.status).toBe(201);

expect(res.body).toHaveProperty('token');

});

});

describe('Tasks', () => {

let authToken;

beforeAll(async () => {

const loginRes = await request(app)

.post('/api/auth/login')

.send({

email: 'test@example.com',

password: 'password123'

});

authToken = loginRes.body.token;

});

test('should create task', async () => {

const res = await request(app)

.post('/api/tasks')

.set('Authorization', `Bearer ${authToken}`)

.send({

title: 'Test Task',

description: 'Test Description'

});

expect(res.status).toBe(201);

});

});

// security/index.js

const helmet = require('helmet');

const rateLimit = require('express-rate-limit');

const securityConfig = {

helmet: helmet({

contentSecurityPolicy: {

directives: {

defaultSrc: ["'self'"],

scriptSrc: ["'self'"],

styleSrc: ["'self'", "'unsafe-inline'"],

imgSrc: ["'self'", 'data:', 'https:'],

connectSrc: ["'self'"]

}

}

}),

rateLimiter: rateLimit({

windowMs: 15 \* 60 \* 1000,

max: 100

}),

cors: {

origin: process.env.ALLOWED\_ORIGINS.split(','),

methods: ['GET', 'POST', 'PUT', 'DELETE'],

allowedHeaders: ['Content-Type', 'Authorization']

}

};

// ai/enhanced-features.js

class EnhancedAI extends AIService {

async predictTaskCompletion(task) {

const features = await this.extractFeatures(task);

return this.timeEstimationModel.predict(features);

}

async suggestSimilarTasks(task) {

const embedding = await this.generateEmbedding(task);

return this.findSimilarTasks(embedding);

}

async generateTaskSummary(task) {

const completion = await this.openai.complete({

prompt: `Summarize this task: ${task.title}\n${task.description}`,

max\_tokens: 100

});

return completion.choices[0].text;

}

}

// export/formats.js

class AdvancedExport {

async exportToPDF(data) {

const doc = new PDFDocument();

doc.fontSize(20).text('Task Report', { align: 'center' });

data.forEach(task => {

doc.fontSize(14).text(task.title)

.fontSize(12).text(task.description);

});

return doc;

}

async exportToExcel(data) {

const workbook = new ExcelJS.Workbook();

const worksheet = workbook.addWorksheet('Tasks');

worksheet.columns = [

{ header: 'Title', key: 'title' },

{ header: 'Status', key: 'status' },

{ header: 'Priority', key: 'priority' }

];

worksheet.addRows(data);

return workbook;

}

async exportToJSON(data) {

return JSON.stringify(data, null, 2);

}

}

// monitoring/analytics.js

class Analytics {

constructor() {

this.metrics = new MetricsCollector();

this.logger = new Winston.Logger();

}

trackUserActivity(userId, action) {

this.metrics.increment(`user.${action}`);

this.logger.info('User activity', { userId, action });

}

async generateReport(timeframe) {

const data = await this.metrics.query(timeframe);

return this.formatReport(data);

}

}