Ministers Network Assessment Enneagram - Source Code

Project Structure

ministers\_network\_assessment/

├── app.py # Main Flask application

├── models.py # Database models

├── extensions.py # Flask extensions

├── static/

│ ├── css/ # Stylesheets

│ │ └── custom.css

│ └── js/ # JavaScript files

│ └── assessment.js

└── templates/ # HTML templates

├── base.html

├── dashboard.html

├── login.html

└── register.html

app.py - Main Application

import os

from datetime import datetime

from flask import Flask, render\_template, request, jsonify, send\_file, flash, redirect, url\_for

from flask\_migrate import Migrate

from flask\_login import LoginManager, login\_user, logout\_user, login\_required, current\_user

import logging

from extensions import db

from models import User, AssessmentResult, DailyReflection, SpiritualQuote # Added SpiritualQuote import

from weasyprint import HTML

from io import BytesIO

# Set up logging

logging.basicConfig(level=logging.DEBUG)

logger = logging.getLogger(\_\_name\_\_)

# Create the app

app = Flask(\_\_name\_\_)

# Configure the app

app.secret\_key = os.environ.get("FLASK\_SECRET\_KEY") or "a secret key"

app.config["SQLALCHEMY\_DATABASE\_URI"] = os.environ.get("DATABASE\_URL")

app.config["SQLALCHEMY\_ENGINE\_OPTIONS"] = {

"pool\_recycle": 300,

"pool\_pre\_ping": True,

}

app.config["SQLALCHEMY\_TRACK\_MODIFICATIONS"] = False

# Initialize extensions

db.init\_app(app)

migrate = Migrate(app, db)

# Initialize Flask-Login

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

login\_manager.login\_message = "Please log in to access this page."

@login\_manager.user\_loader

def load\_user(id):

try:

return db.session.get(User, int(id))

except:

return None

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/register', methods=['GET', 'POST'])

def register():

if request.method == 'POST':

username = request.form.get('username')

email = request.form.get('email')

password = request.form.get('password')

if User.query.filter\_by(username=username).first():

flash('Username already exists')

return redirect(url\_for('register'))

if User.query.filter\_by(email=email).first():

flash('Email already registered')

return redirect(url\_for('register'))

user = User(username=username, email=email)

user.set\_password(password)

db.session.add(user)

try:

db.session.commit()

flash('Registration successful! Please login.')

return redirect(url\_for('login'))

except Exception as e:

logger.error(f"Registration error: {str(e)}")

db.session.rollback()

flash('An error occurred during registration. Please try again.')

return redirect(url\_for('register'))

return render\_template('register.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if current\_user.is\_authenticated:

return redirect(url\_for('dashboard'))

if request.method == 'POST':

username = request.form.get('username')

password = request.form.get('password')

user = User.query.filter\_by(username=username).first()

if user and user.check\_password(password):

login\_user(user)

flash('Logged in successfully.')

next\_page = request.args.get('next')

return redirect(next\_page or url\_for('dashboard'))

flash('Invalid username or password')

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

flash('You have been logged out.')

return redirect(url\_for('index'))

@app.route('/dashboard')

@login\_required

def dashboard():

# Get user's assessment history

try:

assessments = AssessmentResult.query.filter\_by(user\_id=current\_user.id).order\_by(AssessmentResult.created\_at.desc()).all()

except Exception as e:

logger.error(f"Error fetching assessments: {str(e)}")

assessments = []

# Get user's latest reflection

try:

latest\_reflection = DailyReflection.query.filter\_by(user\_id=current\_user.id).order\_by(DailyReflection.created\_at.desc()).first()

except Exception as e:

logger.error(f"Error fetching reflection: {str(e)}")

latest\_reflection = None

# Get a random spiritual quote

try:

quote = SpiritualQuote.query.order\_by(db.func.random()).first()

except Exception as e:

logger.error(f"Error fetching quote: {str(e)}")

quote = None

return render\_template('dashboard.html',

assessments=assessments,

latest\_reflection=latest\_reflection,

quote=quote,

username=current\_user.username)

@app.route('/add\_reflection', methods=['POST'])

@login\_required

def add\_reflection():

try:

data = request.json

reflection = DailyReflection(

user\_id=current\_user.id,

scripture\_reference=data.get('scripture\_reference'),

scripture\_text=data.get('scripture\_text'),

reflection=data.get('reflection'),

prayer\_focus=data.get('prayer\_focus')

)

db.session.add(reflection)

db.session.commit()

return jsonify({'status': 'success'})

except Exception as e:

logger.error(f"Error adding reflection: {str(e)}")

return jsonify({'status': 'error', 'message': str(e)}), 500

@app.route('/save-assessment', methods=['POST'])

@login\_required

def save\_assessment():

try:

data = request.json

logger.debug(f"Received assessment data: {data}")

result = AssessmentResult(

user\_id=current\_user.id,

mental\_health=data['mentalHealth'],

character=data['character'],

doctrinal=data['doctrinal'],

spiritual=data['spiritual'],

relational=data['relational'],

revelatory=data['revelatory'],

presbytery=data['presbytery'],

family\_order=data['familyOrder'],

spiritual\_discipline=data['spiritualDiscipline'],

physical=data['physical']

)

db.session.add(result)

db.session.commit()

logger.info(f"Successfully saved assessment with ID: {result.id}")

return jsonify({

'status': 'success',

'message': 'Assessment saved successfully',

'id': result.id

})

except Exception as e:

logger.error(f"Error saving assessment: {str(e)}")

db.session.rollback()

return jsonify({

'status': 'error',

'message': str(e)

}), 500

@app.route('/generate-pdf', methods=['POST'])

@login\_required

def generate\_pdf():

try:

data = request.json

# Calculate totals

total\_strength = (

data['mentalHealth'] + data['character'] + data['doctrinal'] +

data['spiritual'] + data['relational'] + data['revelatory'] +

data['presbytery']

)

total\_order = data['familyOrder'] + data['spiritualDiscipline'] + data['physical']

total\_score = total\_strength + total\_order

# Render the PDF template

html\_content = render\_template(

'pdf\_report.html',

scores=data,

total\_score=total\_score,

datetime=datetime

)

# Generate PDF

pdf = HTML(string=html\_content).write\_pdf()

# Create a BytesIO object

pdf\_file = BytesIO(pdf)

pdf\_file.seek(0)

return send\_file(

pdf\_file,

mimetype='application/pdf',

as\_attachment=True,

download\_name='assessment\_report.pdf'

)

except Exception as e:

logger.error(f"Error generating PDF: {str(e)}")

return jsonify({

'status': 'error',

'message': str(e)

}), 500

@app.route('/assessment/<int:assessment\_id>')

@login\_required

def get\_assessment(assessment\_id):

try:

assessment = AssessmentResult.query.filter\_by(

id=assessment\_id,

user\_id=current\_user.id

).first\_or\_404()

return jsonify({

'mental\_health': assessment.mental\_health,

'character': assessment.character,

'doctrinal': assessment.doctrinal,

'spiritual': assessment.spiritual,

'relational': assessment.relational,

'revelatory': assessment.revelatory,

'presbytery': assessment.presbytery,

'family\_order': assessment.family\_order,

'spiritual\_discipline': assessment.spiritual\_discipline,

'physical': assessment.physical,

'total\_score': assessment.total\_score,

'created\_at': assessment.created\_at.strftime('%Y-%m-%d %H:%M:%S')

})

except Exception as e:

logger.error(f"Error fetching assessment details: {str(e)}")

return jsonify({'error': 'Assessment not found'}), 404

@app.route('/export-code')

def export\_code():

try:

# Render the HTML template with the code

html\_content = render\_template('code\_export.html')

# Generate PDF from HTML

pdf = HTML(string=html\_content).write\_pdf()

# Create a BytesIO object

pdf\_file = BytesIO(pdf)

pdf\_file.seek(0)

return send\_file(

pdf\_file,

mimetype='application/pdf',

as\_attachment=True,

download\_name='animated\_welcome\_code.pdf'

)

except Exception as e:

logger.error(f"Error generating code export PDF: {str(e)}")

return jsonify({

'status': 'error',

'message': str(e)

}), 500

@app.route('/source-code')

def source\_code():

def read\_file(filepath):

try:

with open(filepath, 'r') as f:

return f.read()

except Exception as e:

logger.error(f"Error reading file {filepath}: {str(e)}")

return f"Error reading file: {str(e)}"

app\_code = read\_file('app.py')

models\_code = read\_file('models.py')

assessment\_js = read\_file('static/js/assessment.js')

custom\_css = read\_file('static/css/custom.css')

migration\_code = read\_file('migrations/versions/initial\_migration.py')

return render\_template('source\_code.html',

app\_code=app\_code,

models\_code=models\_code,

assessment\_js=assessment\_js,

custom\_css=custom\_css,

migration\_code=migration\_code)

if \_\_name\_\_ == '\_\_main\_\_':

with app.app\_context():

db.create\_all()

app.run(host="0.0.0.0", port=5000, debug=True)

Ministers Network Assessment Enneagram - Source Code

Project Structure

ministers\_network\_assessment/

├── app.py # Main Flask application

├── models.py # Database models

├── extensions.py # Flask extensions

├── static/

│ ├── css/ # Stylesheets

│ │ └── custom.css

│ └── js/ # JavaScript files

│ └── assessment.js

└── templates/ # HTML templates

├── base.html

├── dashboard.html

├── login.html

└── register.html

app.py - Main Application

import os

from datetime import datetime

from flask import Flask, render\_template, request, jsonify, send\_file, flash, redirect, url\_for

from flask\_migrate import Migrate

from flask\_login import LoginManager, login\_user, logout\_user, login\_required, current\_user

import logging

from extensions import db

from models import User, AssessmentResult, DailyReflection, SpiritualQuote # Added SpiritualQuote import

from weasyprint import HTML

from io import BytesIO

# Set up logging

logging.basicConfig(level=logging.DEBUG)

logger = logging.getLogger(\_\_name\_\_)

# Create the app

app = Flask(\_\_name\_\_)

# Configure the app

app.secret\_key = os.environ.get("FLASK\_SECRET\_KEY") or "a secret key"

app.config["SQLALCHEMY\_DATABASE\_URI"] = os.environ.get("DATABASE\_URL")

app.config["SQLALCHEMY\_ENGINE\_OPTIONS"] = {

"pool\_recycle": 300,

"pool\_pre\_ping": True,

}

app.config["SQLALCHEMY\_TRACK\_MODIFICATIONS"] = False

# Initialize extensions

db.init\_app(app)

migrate = Migrate(app, db)

# Initialize Flask-Login

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

login\_manager.login\_message = "Please log in to access this page."

@login\_manager.user\_loader

def load\_user(id):

try:

return db.session.get(User, int(id))

except:

return None

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/register', methods=['GET', 'POST'])

def register():

if request.method == 'POST':

username = request.form.get('username')

email = request.form.get('email')

password = request.form.get('password')

if User.query.filter\_by(username=username).first():

flash('Username already exists')

return redirect(url\_for('register'))

if User.query.filter\_by(email=email).first():

flash('Email already registered')

return redirect(url\_for('register'))

user = User(username=username, email=email)

user.set\_password(password)

db.session.add(user)

try:

db.session.commit()

flash('Registration successful! Please login.')

return redirect(url\_for('login'))

except Exception as e:

logger.error(f"Registration error: {str(e)}")

db.session.rollback()

flash('An error occurred during registration. Please try again.')

return redirect(url\_for('register'))

return render\_template('register.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if current\_user.is\_authenticated:

return redirect(url\_for('dashboard'))

if request.method == 'POST':

username = request.form.get('username')

password = request.form.get('password')

user = User.query.filter\_by(username=username).first()

if user and user.check\_password(password):

login\_user(user)

flash('Logged in successfully.')

next\_page = request.args.get('next')

return redirect(next\_page or url\_for('dashboard'))

flash('Invalid username or password')

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

flash('You have been logged out.')

return redirect(url\_for('index'))

@app.route('/dashboard')

@login\_required

def dashboard():

# Get user's assessment history

try:

assessments = AssessmentResult.query.filter\_by(user\_id=current\_user.id).order\_by(AssessmentResult.created\_at.desc()).all()

except Exception as e:

logger.error(f"Error fetching assessments: {str(e)}")

assessments = []

# Get user's latest reflection

try:

latest\_reflection = DailyReflection.query.filter\_by(user\_id=current\_user.id).order\_by(DailyReflection.created\_at.desc()).first()

except Exception as e:

logger.error(f"Error fetching reflection: {str(e)}")

latest\_reflection = None

# Get a random spiritual quote

try:

quote = SpiritualQuote.query.order\_by(db.func.random()).first()

except Exception as e:

logger.error(f"Error fetching quote: {str(e)}")

quote = None

return render\_template('dashboard.html',

assessments=assessments,

latest\_reflection=latest\_reflection,

quote=quote,

username=current\_user.username)

@app.route('/add\_reflection', methods=['POST'])

@login\_required

def add\_reflection():

try:

data = request.json

reflection = DailyReflection(

user\_id=current\_user.id,

scripture\_reference=data.get('scripture\_reference'),

scripture\_text=data.get('scripture\_text'),

reflection=data.get('reflection'),

prayer\_focus=data.get('prayer\_focus')

)

db.session.add(reflection)

db.session.commit()

return jsonify({'status': 'success'})

except Exception as e:

logger.error(f"Error adding reflection: {str(e)}")

return jsonify({'status': 'error', 'message': str(e)}), 500

@app.route('/save-assessment', methods=['POST'])

@login\_required

def save\_assessment():

try:

data = request.json

logger.debug(f"Received assessment data: {data}")

result = AssessmentResult(

user\_id=current\_user.id,

mental\_health=data['mentalHealth'],

character=data['character'],

doctrinal=data['doctrinal'],

spiritual=data['spiritual'],

relational=data['relational'],

revelatory=data['revelatory'],

presbytery=data['presbytery'],

family\_order=data['familyOrder'],

spiritual\_discipline=data['spiritualDiscipline'],

physical=data['physical']

)

db.session.add(result)

db.session.commit()

logger.info(f"Successfully saved assessment with ID: {result.id}")

return jsonify({

'status': 'success',

'message': 'Assessment saved successfully',

'id': result.id

})

except Exception as e:

logger.error(f"Error saving assessment: {str(e)}")

db.session.rollback()

return jsonify({

'status': 'error',

'message': str(e)

}), 500

@app.route('/generate-pdf', methods=['POST'])

@login\_required

def generate\_pdf():

try:

data = request.json

# Calculate totals

total\_strength = (

data['mentalHealth'] + data['character'] + data['doctrinal'] +

data['spiritual'] + data['relational'] + data['revelatory'] +

data['presbytery']

)

total\_order = data['familyOrder'] + data['spiritualDiscipline'] + data['physical']

total\_score = total\_strength + total\_order

# Render the PDF template

html\_content = render\_template(

'pdf\_report.html',

scores=data,

total\_score=total\_score,

datetime=datetime

)

# Generate PDF

pdf = HTML(string=html\_content).write\_pdf()

# Create a BytesIO object

pdf\_file = BytesIO(pdf)

pdf\_file.seek(0)

return send\_file(

pdf\_file,

mimetype='application/pdf',

as\_attachment=True,

download\_name='assessment\_report.pdf'

)

except Exception as e:

logger.error(f"Error generating PDF: {str(e)}")

return jsonify({

'status': 'error',

'message': str(e)

}), 500

@app.route('/assessment/<int:assessment\_id>')

@login\_required

def get\_assessment(assessment\_id):

try:

assessment = AssessmentResult.query.filter\_by(

id=assessment\_id,

user\_id=current\_user.id

).first\_or\_404()

return jsonify({

'mental\_health': assessment.mental\_health,

'character': assessment.character,

'doctrinal': assessment.doctrinal,

'spiritual': assessment.spiritual,

'relational': assessment.relational,

'revelatory': assessment.revelatory,

'presbytery': assessment.presbytery,

'family\_order': assessment.family\_order,

'spiritual\_discipline': assessment.spiritual\_discipline,

'physical': assessment.physical,

'total\_score': assessment.total\_score,

'created\_at': assessment.created\_at.strftime('%Y-%m-%d %H:%M:%S')

})

except Exception as e:

logger.error(f"Error fetching assessment details: {str(e)}")

return jsonify({'error': 'Assessment not found'}), 404

@app.route('/export-code')

def export\_code():

try:

# Render the HTML template with the code

html\_content = render\_template('code\_export.html')

# Generate PDF from HTML

pdf = HTML(string=html\_content).write\_pdf()

# Create a BytesIO object

pdf\_file = BytesIO(pdf)

pdf\_file.seek(0)

return send\_file(

pdf\_file,

mimetype='application/pdf',

as\_attachment=True,

download\_name='animated\_welcome\_code.pdf'

)

except Exception as e:

logger.error(f"Error generating code export PDF: {str(e)}")

return jsonify({

'status': 'error',

'message': str(e)

}), 500

@app.route('/source-code')

def source\_code():

def read\_file(filepath):

try:

with open(filepath, 'r') as f:

return f.read()

except Exception as e:

logger.error(f"Error reading file {filepath}: {str(e)}")

return f"Error reading file: {str(e)}"

app\_code = read\_file('app.py')

models\_code = read\_file('models.py')

assessment\_js = read\_file('static/js/assessment.js')

custom\_css = read\_file('static/css/custom.css')

migration\_code = read\_file('migrations/versions/initial\_migration.py')

return render\_template('source\_code.html',

app\_code=app\_code,

models\_code=models\_code,

assessment\_js=assessment\_js,

custom\_css=custom\_css,

migration\_code=migration\_code)

if \_\_name\_\_ == '\_\_main\_\_':

with app.app\_context():

db.create\_all()

app.run(host="0.0.0.0", port=5000, debug=True)

models.py - Database Models

from datetime import datetime

from extensions import db

from flask\_login import UserMixin

from werkzeug.security import generate\_password\_hash, check\_password\_hash

class User(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(64), unique=True, nullable=False)

email = db.Column(db.String(120), unique=True, nullable=False)

password\_hash = db.Column(db.String(256))

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

assessments = db.relationship('AssessmentResult', backref='user', lazy=True)

def set\_password(self, password):

self.password\_hash = generate\_password\_hash(password)

def check\_password(self, password):

return check\_password\_hash(self.password\_hash, password)

def \_\_repr\_\_(self):

return f'<User {self.username}>'

class SpiritualQuote(db.Model):

id = db.Column(db.Integer, primary\_key=True)

text = db.Column(db.Text, nullable=False)

reference = db.Column(db.String(100), nullable=False)

category = db.Column(db.String(50), nullable=False)

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

def \_\_repr\_\_(self):

return f'<SpiritualQuote {self.reference}>'

class AssessmentResult(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

# Strength Metrics (1-10)

mental\_health = db.Column(db.Integer, nullable=False)

character = db.Column(db.Integer, nullable=False)

doctrinal = db.Column(db.Integer, nullable=False)

spiritual = db.Column(db.Integer, nullable=False)

relational = db.Column(db.Integer, nullable=False)

revelatory = db.Column(db.Integer, nullable=False)

presbytery = db.Column(db.Integer, nullable=False)

# Order Metrics (1-5)

family\_order = db.Column(db.Integer, nullable=False)

spiritual\_discipline = db.Column(db.Integer, nullable=False)

physical = db.Column(db.Integer, nullable=False)

# Calculated totals

total\_strength = db.Column(db.Integer)

total\_order = db.Column(db.Integer)

total\_score = db.Column(db.Integer)

def \_\_init\_\_(self, \*\*kwargs):

super(AssessmentResult, self).\_\_init\_\_(\*\*kwargs)

self.calculate\_totals()

def calculate\_totals(self):

self.total\_strength = (

self.mental\_health + self.character + self.doctrinal +

self.spiritual + self.relational + self.revelatory +

self.presbytery

)

self.total\_order = (

self.family\_order + self.spiritual\_discipline + self.physical

)

self.total\_score = self.total\_strength + self.total\_order

class DailyReflection(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

date = db.Column(db.Date, default=datetime.utcnow().date)

scripture\_reference = db.Column(db.String(100))

scripture\_text = db.Column(db.Text)

reflection = db.Column(db.Text)

prayer\_focus = db.Column(db.Text)

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

static/js/assessment.js - Assessment Logic

async function generateReport() {

// Get all input values

const scores = {

mentalHealth: getInputValue("mentalHealth"),

character: getInputValue("character"),

doctrinal: getInputValue("doctrinal"),

spiritual: getInputValue("spiritual"),

relational: getInputValue("relational"),

revelatory: getInputValue("revelatory"),

presbytery: getInputValue("presbytery"),

familyOrder: getInputValue("familyOrder"),

spiritualDiscipline: getInputValue("spiritualDiscipline"),

physical: getInputValue("physical")

};

// Validate inputs

const emptyFields = Object.entries(scores).filter(([\_, value]) => !value);

if (emptyFields.length > 0) {

alert("Please fill in all fields before generating the report.");

return;

}

// Save assessment to database

try {

const response = await fetch('/save-assessment', {

method: 'POST',

headers: {

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

},

body: JSON.stringify(scores)

});

const result = await response.json();

if (result.status === 'success') {

// Redirect to dashboard after successful save

window.location.href = '/dashboard';

return;

} else {

console.error('Error saving assessment:', result.message);

alert('Error saving assessment. Please try again.');

return;

}

} catch (error) {

console.error('Error saving assessment:', error);

alert('Error saving assessment. Please try again.');

return;

}

// Calculate totals

const totalStrength = scores.mentalHealth + scores.character + scores.doctrinal +

scores.spiritual + scores.relational + scores.revelatory + scores.presbytery;

const totalOrder = scores.familyOrder + scores.spiritualDiscipline + scores.physical;

const totalScore = totalStrength + totalOrder;

// Generate chart

const ctx = document.getElementById('enneagramChart').getContext('2d');

if (window.myChart instanceof Chart) {

window.myChart.destroy();

}

window.myChart = new Chart(ctx, {

type: 'radar',

data: {

labels: [

'Mental Health', 'Character', 'Doctrinal',

'Spiritual', 'Relational', 'Revelatory',

'Presbytery', 'Family Order', 'Spiritual Discipline', 'Physical'

],

datasets: [{

label: 'Your Assessment',

data: Object.values(scores),

fill: true,

backgroundColor: 'rgba(54, 162, 235, 0.2)',

borderColor: 'rgb(54, 162, 235)',

pointBackgroundColor: 'rgb(54, 162, 235)',

pointBorderColor: '#fff',

pointHoverBackgroundColor: '#fff',

pointHoverBorderColor: 'rgb(54, 162, 235)'

}]

},

options: {

elements: {

line: {

borderWidth: 3

}

},

scales: {

r: {

angleLines: {

display: true

},

suggestedMin: 0,

suggestedMax: 10

}

}

}

});

// Generate and download PDF

try {

const pdfResponse = await fetch('/generate-pdf', {

method: 'POST',

headers: {

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

},

body: JSON.stringify(scores)

});

if (pdfResponse.ok) {

const blob = await pdfResponse.blob();

const url = window.URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = 'assessment\_report.pdf';

document.body.appendChild(a);

a.click();

window.URL.revokeObjectURL(url);

a.remove();

} else {

console.error('Error generating PDF');

}

} catch (error) {

console.error('Error generating PDF:', error);

}

// Show result alert

let interpretation = "";

if (totalScore <= 33) {

interpretation = "Weak: Significant growth needed in multiple areas.";

} else if (totalScore <= 66) {

interpretation = "Moderate: Balanced but requires attention in specific areas.";

} else {

interpretation = "Healthy: Strong alignment and maturity in most areas. Keep growing!";

}

alert(`Your Total Score: ${totalScore} (Maximum: 80)\n\n${interpretation}`);

}

function getInputValue(id) {

const value = parseInt(document.getElementById(id).value) || 0;

if (value < 1) return 1;

if (id.includes('family') || id.includes('spiritual') || id === 'physical') {

return Math.min(value, 5);

}

return Math.min(value, 10);

}

// Add input validation

document.addEventListener('DOMContentLoaded', function() {

const inputs = document.querySelectorAll('input[type="number"]');

inputs.forEach(input => {

input.addEventListener('input', function() {

let max = 10;

if (this.id.includes('family') || this.id.includes('spiritual') || this.id === 'physical') {

max = 5;

}

let value = parseInt(this.value);

if (value > max) this.value = max;

if (value < 1 || isNaN(value)) this.value = 1;

});

});

});

static/css/custom.css - Custom Styles

/\* Custom styles to complement Bootstrap \*/

.interpretation ul {

padding-left: 0;

}

.score-interpretation ul {

list-style-type: none;

padding-left: 0;

}

.form-control:focus {

border-color: var(--bs-primary);

box-shadow: 0 0 0 0.25rem rgba(var(--bs-primary-rgb), 0.25);

}

.card {

border-radius: 0.5rem;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

}

.alert {

border-radius: 0.5rem;

}

/\* SMART Goals Styling \*/

.smart-goals .accordion-button:not(.collapsed) {

background-color: var(--bs-primary);

color: var(--bs-white);

}

.smart-goals .accordion-button:focus {

box-shadow: 0 0 0 0.25rem rgba(var(--bs-primary-rgb), 0.25);

}

.smart-goals ol {

padding-left: 1.2rem;

}

.smart-goals ol li {

margin-bottom: 0.75rem;

}

.smart-goals .goal-section {

margin-bottom: 2rem;

padding: 1rem;

border-left: 4px solid var(--bs-primary);

background-color: var(--bs-dark);

}

/\* Ensure proper spacing for inputs \*/

.mb-3 input {

max-width: 100%;

}

/\* Make canvas responsive \*/

canvas {

max-width: 100%;

height: auto;

}

/\* Improve small text readability \*/

.small {

opacity: 0.8;

}

/\* Improve button hover state \*/

.btn-primary:hover {

transform: translateY(-1px);

transition: transform 0.2s ease;

}

/\* Welcome Message Animation \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

.welcome-message {

animation: fadeIn 1s ease-out;

}

.spiritual-quote {

position: relative;

padding: 1.5rem;

margin-bottom: 2rem;

border-radius: 0.5rem;

background: var(--bs-dark);

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

animation: fadeIn 1.5s ease-out;

}

.spiritual-quote::before {

content: '"';

position: absolute;

top: -1rem;

left: 1rem;

font-size: 4rem;

color: var(--bs-primary);

opacity: 0.2;

}

Initial Database Migration

"""Initial migration

Revision ID: initial\_migration

Revises:

Create Date: 2025-01-29 05:37:33.000000

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = 'initial\_migration'

down\_revision = None

branch\_labels = None

depends\_on = None

def upgrade():

# Create User table

op.create\_table('user',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('username', sa.String(length=64), nullable=False),

sa.Column('email', sa.String(length=120), nullable=False),

sa.Column('password\_hash', sa.String(length=256), nullable=True),

sa.Column('created\_at', sa.DateTime(), nullable=True),

sa.PrimaryKeyConstraint('id'),

sa.UniqueConstraint('email'),

sa.UniqueConstraint('username')

)

# Create AssessmentResult table

op.create\_table('assessment\_result',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('user\_id', sa.Integer(), nullable=False),

sa.Column('created\_at', sa.DateTime(), nullable=True),

sa.Column('scripture\_reflection', sa.Text(), nullable=True),

sa.Column('prayer\_notes', sa.Text(), nullable=True),

sa.Column('mental\_health', sa.Integer(), nullable=False),

sa.Column('character', sa.Integer(), nullable=False),

sa.Column('doctrinal', sa.Integer(), nullable=False),

sa.Column('spiritual', sa.Integer(), nullable=False),

sa.Column('relational', sa.Integer(), nullable=False),

sa.Column('revelatory', sa.Integer(), nullable=False),

sa.Column('presbytery', sa.Integer(), nullable=False),

sa.Column('family\_order', sa.Integer(), nullable=False),

sa.Column('spiritual\_discipline', sa.Integer(), nullable=False),

sa.Column('physical', sa.Integer(), nullable=False),

sa.Column('total\_strength', sa.Integer(), nullable=True),

sa.Column('total\_order', sa.Integer(), nullable=True),

sa.Column('total\_score', sa.Integer(), nullable=True),

sa.ForeignKeyConstraint(['user\_id'], ['user.id'], ),

sa.PrimaryKeyConstraint('id')

)

# Create DailyReflection table

op.create\_table('daily\_reflection',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('user\_id', sa.Integer(), nullable=False),

sa.Column('date', sa.Date(), nullable=True),

sa.Column('scripture\_reference', sa.String(length=100), nullable=True),

sa.Column('scripture\_text', sa.Text(), nullable=True),

sa.Column('reflection', sa.Text(), nullable=True),

sa.Column('prayer\_focus', sa.Text(), nullable=True),

sa.Column('created\_at', sa.DateTime(), nullable=True),

sa.ForeignKeyConstraint(['user\_id'], ['user.id'], ),

sa.PrimaryKeyConstraint('id')

)

def downgrade():

op.drop\_table('daily\_reflection')

op.drop\_table('assessment\_result')

op.drop\_table('user')