Table of contents

Project Overview

This document provides a high-level overview of the ResuMate project, including its vision, goals, and scope.

Vision

ResuMate aims to revolutionize the job application process by leveraging AI and ML technologies to help users create, analyze, and optimize their Résumés and CVs. The platform focuses on enhancing the quality of Résumés, making them more appealing to both Applicant Tracking Systems (ATS) and human recruiters.

Goals

The primary goals of the ResuMate project are to:

- Develop a user-friendly interface for easy Résumé and CV management.
- Implement advanced natural language processing (NLP) techniques for accurate content analysis.
- Provide constructive feedback to help users improve their documents.
- Offer customizable templates to enhance the visual appeal of Résumés and CVs.
- Ensure data privacy and security in compliance with relevant regulations.
- Facilitate continuous learning to improve the platform's feedback mechanisms.
- Deliver a scalable solution capable of handling a growing user base and feature set.

Scope

Core Functionalities for MVP

The Minimum Viable Product (MVP) will include the following core functionalities:

- **Document Upload**: Enable users to upload their Résumés or CVs in various formats (PDF, DOCX, etc.).

- **Content Analysis**: Utilize AI to analyze the document's content for clarity, relevance, and impact.
- **Feedback and Suggestions**: Provide AI-generated feedback on how to improve the document.
- **Template Selection**: Allow users to choose from three distinct templates for Résumé or CV creation.
- **Sample Generation**: Offer the ability to auto-generate a sample Résumé or CV based on user-provided information.
- **Customization**: Allow users to customize sections such as work experience, education, and skills.
- **Download**: Enable users to download the updated Résumé or CV in their chosen format.

Technologies and Frameworks

ResuMate uses a robust stack of technologies and frameworks to deliver its functionalities seamlessly:

- **Backend**: Python with Flask or Django for server-side logic.
- **AI/ML**: TensorFlow or PyTorch for ML models, spaCy for NLP tasks.
- **Frontend**: Vue.js for an interactive UI, HTML/CSS for structure and style.
- **Database**: SQL database to store user data and document information.
- **Cloud Hosting**: AWS or Azure for hosting the application and database.
- **Version Control**: Git for source code management, hosted on GitHub or GitLab.

Project Timeline

The project is divided into several key phases, each with specific tasks and milestones:

1. **Project Planning**: Define project scope and MVP, create user stories and epics, develop project timeline and sprints.

2. **Environment Setup**: Set up development and testing environments, configure version control and project management tools.
3. **Data Collection**: Gather Résumé datasets, define data storage and retrieval mechanisms.
4. **Backend Development**: Implement Résumé parsing, develop content analysis algorithms, create feedback generation logic.
5. **Frontend Development**: Design user interface, develop interactive components, integrate with backend services.
6. **Machine Learning Integration**: Train NLP models, integrate ML models with backend, validate model performance.
7. **Testing**: Write unit tests for individual components, conduct integration testing, perform user acceptance testing.
8. **Documentation**: Document code and system architecture, create developer and user manuals.
9. **Deployment**: Prepare deployment strategy, deploy application to production, monitor application performance.
10. **Project Review and Retrospective**: Review project deliverables, conduct sprint retrospectives, plan for future iterations.

Future Expansion

As ResuMate establishes itself in the software development sector, the vision extends to:

- **Strategic Category Expansion**: Introduce new job categories, prioritizing fields where AI and ML technologies can have the most impact.
- **Adaptive Learning Algorithms**: Evolve the platform to learn from a broader spectrum of professional terminologies and success metrics.
- **Collaborative Ecosystem Development**: Collaborate with industry experts and hiring professionals to keep insights and templates up to date.

Conclusion

The ResuMate project is set to become a crucial tool in the job market, offering unmatched support to jobseekers in creating Résumés and CVs that resonate with employers. With a clear vision and a strong technical strategy, ResuMate aims to transform the job application landscape and empower users to succeed in their career pursuits.

Objectives

This document outlines the key objectives of the ResuMate project. These objectives guide the development process and ensure that the final product meets the needs of users and stakeholders.

Primary Objectives

1. Develop a User-Friendly Interface

Create an intuitive and accessible platform for users to easily upload and manage their resumes and CVs.

- **User Experience**: Ensure that the interface is straightforward, clean, and uncomplicated to navigate.
- **Accessibility**: Design the platform to be accessible to users with disabilities, adhering to WCAG guidelines.

2. Implement Advanced NLP Techniques

Use natural language processing (NLP) to accurately parse and analyze resume and CV content.

- **Parsing Accuracy**: Develop algorithms to correctly identify and extract relevant information from resumes.
- **Analysis**: Analyze the content for clarity, relevance, and impact, providing meaningful feedback.

3. Provide Constructive Feedback

Generate meaningful suggestions that help users improve the content and presentation of their resumes and CVs.

- **Content Improvement**: Offer suggestions to enhance the readability and effectiveness of the résumé.

- **ATS Compatibility**: Ensure the résumé is optimized for Applicant Tracking Systems (ATS).

4. Offer Customizable Templates

Allow users to apply professional templates that enhance the visual appeal of their resumes and CVs and reflect their unique skills and experiences.

- **Template Variety**: Provides a selection of professional, modern, and creative templates.
- **Customization**: Enable users to customize sections, fonts, and colors to suit their preferences.

5. Ensure Data Privacy and Security

Maintain the highest standards of data protection to safeguard user information, adhering to GDPR, CCPA, or other relevant data privacy regulations.

- **Data Encryption**: Implement encryption for data storage and transmission.
- **Privacy Compliance**: Ensure compliance with all relevant data protection regulations.

6. Facilitate Continuous Learning

Incorporate a machine learning model that improves its feedback over time through user interactions, ensuring the platform evolves to meet user needs.

- **Model Training**: Continuously train the ML model with new data to improve accuracy.
- **User Feedback**: Integrate user feedback to enhance the model's performance.

7. Deliver a Scalable Solution

Build the application with scalability in mind to accommodate a growing user base and feature set, ensuring seamless performance as the platform expands.

- **Scalability**: Design the architecture to handle increasing amounts of data and users.
- **Performance Optimization**: Continuously monitor and optimize the application's

performance.

Secondary Objectives

1. Expand Job Categories

After establishing a strong presence in the software development sector, expand the platform to cover other job categories.

- **Market Research**: Conduct research to identify the most impactful job categories to target.
- **Algorithm Adaptation**: Adapt algorithms to suit the specific needs of different job sectors.

2. Enhance User Engagement

Develop features that increase user engagement and satisfaction.

- **User Insights**: Provide insights and analytics to help users understand how their resumes are performing.
- **Gamification**: Implement gamification elements to encourage users to improve their resumes.

3. Build Collaborative Ecosystem

Collaborate with industry experts and hiring professionals to ensure that the platform remains up to date with the latest trends and best practices.

- **Expert Collaboration**: Work with HR professionals and recruiters to gather insights.
- **Community Building**: Foster a community of users who can share tips and feedback.

Conclusion

These objectives guide the ResuMate project to ensure the development of a high-quality, user-centric platform that helps jobseekers create effective resumes and CVs. By

focusing on these goals, the ResuMate team aims to revolutionize the job application process and empower users to succeed in their career pursuits.

Technologies Used

This document provides an overview of the technologies and frameworks used in the ResuMate project.

Backend Technologies

Python

Python is the core programming language used for the backend of the ResuMate project.

Flask

Flask is a lightweight web framework for Python that is used to handle the server-side logic of the application.

SQLAlchemy

SQLAlchemy is an ORM (Object-Relational Mapping) library for Python used to interact with the SQL database.

TensorFlow / PyTorch

These machine learning frameworks are used for building and training AI models.

SpaCy

SpaCy is an NLP (Natural Language Processing) library used for parsing and analyzing resume content.

Frontend Technologies

Vue.js

Vue.js is a progressive JavaScript framework used to build the user interface of the application.

HTML/CSS

HTML and CSS are used for structuring and styling the web pages.

JavaScript

JavaScript is used for adding interactivity to the frontend components.

Database

SQL Database

An SQL database is used to store user data and document information.

Cloud Hosting

AWS / Azure

AWS or Azure are used for hosting the application and database in the cloud.

Version Control

Git

Git is used for version control and tracking changes in the source code.

GitHub / GitLab

GitHub or GitLab are used for hosting the remote repository and collaboration.

Development Environment

PyCharm

PyCharm is an IDE (Integrated Development Environment) used for Python development.

Virtual Environment

A virtual environment is used to manage project dependencies separately from the global Python environment.

Additional Tools

Jupyter Notebook

Jupyter Notebook is used for interactive development and testing of machine learning models.

Pandas / NumPy

Pandas and NumPy are libraries used for data manipulation and numerical computations.

Sphinx

Sphinx is used for generating documentation from the Python code.

Markdown

Markdown is used for writing README and other documentation files.

Conclusion

The ResuMate project leverages a comprehensive stack of technologies and frameworks to deliver its functionalities. By using these technologies, the project ensures a robust, scalable, and efficient development process.

Environment Setup

Install Python

Ensure Python is installed on your system. Download it from the official website (https://www.python.org/downloads/) if needed.

Set Up a Virtual Environment

Create a virtual environment to manage project dependencies separately from the global Python environment.

Command to create a virtual environment:

python -m venv .venv

Command to activate the virtual environment:

- **Windows**: `.\.venv\Scripts\Activate.ps1`
- **MacOS/Linux**: `source .venv/bin/activate`

Verify the Virtual Environment

After activation, your command prompt or terminal should indicate that you are working within the virtual environment (usually by showing the name of the virtual environment before the prompt).

Install Project Dependencies

With the virtual environment activated, install the necessary project dependencies. Ensure you have a `requirements.txt` file in your project root directory. Install the dependencies using the following command: `pip install -r requirements.txt`

Additional Tools

Install Git

Ensure Git is installed for version control. You can download it from the official Git

website (https://git-scm.com/). Verify the installation by running: `git --version`

Set Up Your IDE

Use an Integrated Development Environment (IDE) like PyCharm or Visual Studio Code for a more efficient development workflow. Here are basic steps for setting up PyCharm:

- **Install PyCharm**: Download and install PyCharm from the JetBrains website (https://www.jetbrains.com/pycharm/download/).
- **Open the Project**: Open PyCharm and select "Open" to navigate to your project directory.
- **Configure the Python Interpreter**: Go to `File > Settings > Project: [Your Project] > Python Interpreter`
- and select the interpreter from your `.venv` directory.

Final Check

Ensure all setups are complete by running the Flask development server. Navigate to your project directory and run: `python run.py`

Visit `http://127.0.0.1:5000/` in your web browser to check if the application is running correctly.

Dependencies Installation

This document provides instructions for installing and managing the project dependencies for ResuMate.

Setting Up the Virtual Environment

First, ensure you have set up the virtual environment as described in the Environment Setup section. If not, follow these steps:

Set Up Virtual Environment

1. Open your terminal or command prompt and navigate to the root directory of your project.
2. Create a virtual environment by running the following command:
   - `python -m venv .venv`
3. Activate the virtual environment:
   - **Windows:**
     - `.\.venv\Scripts\Activate.ps1`
   - **macOS/Linux:**
     - `source .venv/bin/activate`

Installing Dependencies

With the virtual environment activated, you can install the project dependencies.

Install Dependencies

1. Ensure you have a `requirements.txt` file in your project root directory. If not, create one and add the required dependencies.
2. Install the dependencies using the following command:

- `pip install -r requirements.txt`

Managing Dependencies

To manage dependencies effectively, follow these guidelines:

Adding New Dependencies

Add New Dependencies

1. When you need to add a new package to your project, use the following command:
   - `pip install <package_name>`
2. After installing the new package, update the `requirements.txt` file:
   - `pip freeze > requirements.txt`

Updating Dependencies

Update Dependencies

1. To update a specific package, use the following command:
   - `pip install --upgrade <package_name>`
2. After updating, remember to update the `requirements.txt` file:
   - `pip freeze > requirements.txt`

Removing Dependencies

Remove Dependencies

1. If you need to remove a package, use the following command:
   - `pip uninstall <package_name>`
2. After removing the package, update the `requirements.txt` file:
   - `pip freeze > requirements.txt`

Checking Dependency Versions

Check Dependency Versions

- To check the installed versions of all dependencies, use the following command:
  - `pip list`

New Dependencies for User Authentication

To set up user authentication, we added the following dependencies:

- `Flask-Login`: For user session management.
- `Flask-Bcrypt`: For hashing passwords.

Installing New Dependencies

Install New Dependencies

1. To install the new dependencies, use the following command:
   - `pip install flask-login flask-bcrypt`
2. After installing the new packages, update the `requirements.txt` file:
   - `pip freeze > requirements.txt`

Common Issues and Troubleshooting

Issue: Package Installation Error

Resolve Package Installation Error

- If you encounter an error during package installation, try the following steps:
  - Ensure your virtual environment is activated.
  - Verify the package name and version.

- Check your internet connection.
- If the issue persists, consult the package documentation or community forums.

Issue: Dependency Conflicts

Resolve Dependency Conflicts

- If there are conflicts between package versions, consider using a tool like `pip-tools` to manage dependencies and resolve conflicts:
  - `pip install pip-tools`
  - `pip-compile`

Conclusion

Managing project dependencies is crucial for maintaining a stable and functional development environment. By following these guidelines, you can ensure that all necessary packages are installed and up to date, making it easier to collaborate and develop the ResuMate project.

Project Structure

This document provides an overview of the project's directory structure, explaining the purpose of each directory and file.

Root Directory

The root directory contains the main project files and folders.

Create ResuMate Project Structure

-
  - `.idea/`
  - `.venv/`
  - `app/`
    - `__init__.py`
    - `routes.py`
  - `static/`
  - `templates/`
    - `upload.html`
  - `test/`
    - `__init__.py`
    - `test_app.py`
  - `uploads/`
  - `Writerside/`
    - `images/`
    - `topics/`
      - `Implementation Details/`

        - Introduction/
        - Next Steps/
        - Project Setup/
        - Testing/
        - Sprint Review and Retrospective/
    - config.py
    - requirements.txt
    - run.py

Directory and File Descriptions

.idea/

Contains IDE-specific settings and project configuration files.

.venv/

Contains the virtual environment for the project, including installed dependencies.

app/

The main application directory containing the core Flask application files.

- **init .py**: Initializes the Flask application.
- **routes.py**: Contains the routes for handling different web requests.

static/

Contains static files such as CSS, JavaScript, and images.

templates/

Contains HTML templates for rendering web pages.

- **upload.html**: Template for the file upload page.

test/

Contains unit tests for the application.

- **init .py**: Initializes the test module.
- **test_app.py**: Contains unit tests for the Flask application.

uploads/

Directory for storing uploaded files.

Writerside/

Contains documentation files and images for the Writerside documentation tool.

- **images/**: Contains images used in the documentation.
- **topics/**: Contains markdown files for different documentation topics.
  - **Implementation Details/**: Detailed implementation guides.
  - **Introduction/**: Introduction to the project.
  - **Next Steps/**: Future plans and features.
  - **Project Setup/**: Setup instructions.
  - **Testing/**: Testing guides.
  - **Sprint Review and Retrospective/**: Review and retrospective notes.

config.py

Configuration file for the Flask application.

requirements.txt

Lists the project dependencies that need to be installed.

run.py

Script to run the Flask development server.

Conclusion

The project structure is organized to separate different concerns, making the project easy to navigate and maintain. Each directory and file has a specific role, contributing to the overall functionality and organization of the project.

Flask Application Setup

This document provides instructions for setting up a Flask application with essential extensions.

Setting Up the Flask Application

First, ensure you have set up the virtual environment as described in the Environment Setup section.

Set Up Flask Application

- Install Flask by running the following command:

  pip install Flask

Install Dependencies

Ensure all necessary packages are installed in your virtual environment.

Install Dependencies

1. Open your terminal or command prompt.
2. Run the following command to install the required packages:
   - `pip install flask flask-sqlalchemy flask-migrate flask-bcrypt flask-login`

Create the Application Structure

Create the necessary directories and files for your Flask application.

Create Application Structure

- In your project root directory, create the following structure:

  project/ │ ├── app/ │ ├── init.py │ ├── models.py │ ├── routes.py │ ├── forms.py │ └── templates/ │ ├── base.html │ ├── index.html │ ├── create_user.html │ ├── edit_user.html │ ├── list_users.html │ ├── login.html │ └── user_detail.html │ ├── migrations/ │ ├── static/ │ └── style.css │ ├──

instance/ │ └── resumate.db │ ├── .venv/ │ ├── config.py │ ├── wsgi.py │ └── requirements.txt

Initialize Flask Application

Initialize the Flask application in `__init__.py`.

Initialize Flask Application

- Add the following code to your `app/__init__.py`:

  from flask import Flask app = Flask(__name__) from app import routes

Update __init__.py in the app directory:

Initialize Flask Application

- Add the following code to your `__init__.py`:

  ```python
  from flask import Flask
  from flask_sqlalchemy import SQLAlchemy
  from flask_migrate import Migrate
  from flask_bcrypt import Bcrypt
  from flask_login import LoginManager
  from config import Config

  db = SQLAlchemy()
  migrate = Migrate()
  bcrypt = Bcrypt()
  login_manager = LoginManager()
  login_manager.login_view = 'login'
  login_manager.login_message_category = 'info'

  def create_app(config_class=Config):
      app = Flask(__name__, template_folder='../templates')
      app.config.from_object(config_class)
  ```

```
    db.init_app(app)
    migrate.init_app(app, db)
    bcrypt.init_app(app)
    login_manager.init_app(app)

    from app import routes  # Import routes module
    routes.init_routes(app)  # Initialize routes

    from app.models import User  # Import User model for login
management

    @login_manager.user_loader
    def load_user(user_id):
        return User.query.get(int(user_id))

    return app
```

Create Routes

Define your application routes in `routes.py`.

Create Routes

- ```python
  from flask import render_template, url_for, flash, redirect, request
  from app import app, db, bcrypt
  from app.forms import RegistrationForm, LoginForm
  from app.models import User
  from flask_login import login_user, current_user, logout_user,
  login_required

  def init_routes(app):
      @app.route("/")
      @app.route("/index")
      def index():
          return render_template('index.html', title='Home')

```
    @app.route("/register", methods=['GET', 'POST'])
    def register():
        if current_user.is_authenticated:
            return redirect(url_for('index'))
        form = RegistrationForm()
        if form.validate_on_submit():
            hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
            user = User(username=form.username.data,
email=form.email.data, password_hash=hashed_password)
            db.session.add(user)
            db.session.commit()
            flash('Your account has been created!', 'success')
            return redirect(url_for('login'))
        return render_template('register.html', title='Register',
form=form)

    @app.route("/login", methods=['GET', 'POST'])
    def login():
        if current_user.is_authenticated:
            return redirect(url_for('index'))
        form = LoginForm()
        if form.validate_on_submit():
            user =
User.query.filter_by(email=form.email.data).first()
            if user and
bcrypt.check_password_hash(user.password_hash, form.password.data):
                login_user(user, remember=form.remember.data)
                next_page = request.args.get('next')
                return redirect(next_page) if next_page else
redirect(url_for('index'))
            else:
                flash('Login unsuccessful. Please check email and
password', 'danger')
        return render_template('login.html', title='Login',
form=form)
```

```
    @app.route("/logout")
    def logout():
        logout_user()
        return redirect(url_for('index'))

    @app.route("/users")
    @login_required
    def users():
        users = User.query.all()
        return render_template('users.html', users=users)
```

Run the Application

Run the Flask application using `run.py`.

Run Application

1. Create a `run.py` file with the following code:

   from app import app if __name__ == '__main__': app.run(debug=True)

2. Run the application:

   python run.py

Conclusion

Setting up a Flask application involves creating the application structure, initializing the application, defining routes, and running the application. By following these steps, you will have a basic Flask application up and running.

Résumé Parsing with SpaCy

This document provides instructions for setting up Résumé parsing using SpaCy in your Flask application.

Install SpaCy and Language Model

First, ensure SpaCy and the necessary language model are installed in your virtual environment.

Install SpaCy and Language Model

1. Open your terminal or command prompt.
2. Run the following command to install SpaCy:

```
pip install spacy
```

Download the English language model:

```
python -m spacy download en_core_web_sm
```

Initialize SpaCy in Your Application

Set up SpaCy in your Flask application by initializing it in a new file, such as `nlp.py`, in the `app` directory.

Example nlp.py:

Initialize SpaCy

- Create a new file `nlp.py` in the `app` directory and add the following code:

```
import spacy
```

- Load the pre-trained SpaCy model

```
nlp = spacy.load('en_core_web_sm')
```

Create Résumé Parsing Function

Define a function to parse Résumés using SpaCy in the `nlp.py` file.

Example Résumé Parsing Function:

Create Résumé Parsing Function

- Add the following code to your `nlp.py`:

```
from collections import defaultdict

def parse_ Résumé( Résumé_text):
doc = nlp( Résumé_text)
parsed_ Résumé = defaultdict(str)

# Extracting entities
for ent in doc.ents:
    parsed_ Résumé[ent.label_] += ent.text + ' '

return parsed_ Résumé
```

Integrate Résumé Parsing with Flask

Update your `routes.py` file to include a route for parsing Résumés.

Example routes.py:

Integrate Résumé Parsing with Flask

- Add the following code to your `routes.py`:

```
from flask import request, jsonify
from .nlp import parse_ Résumé

@app.route('/parse_ Résumé', methods=['POST'])
```

```
def parse_ Résumé_route():
 Résumé_text = request.json.get(' Résumé_text', '')
parsed_ Résumé = parse_ Résumé( Résumé_text)
return jsonify(parsed_ Résumé)
```

Test the Résumé Parsing Endpoint

Create a simple client to test the Résumé parsing endpoint.

Example Client Code:

Test Résumé Parsing Endpoint

- Create a new file `test_ Résumé_parsing.py` in the root directory and add the following code:

  `import requests`

  url = 'http://127.0.0.1:5000/parse_ Résumé' data = {' Résumé_text': 'Experienced Python developer with a background in machine learning and data analysis.'}

  response = requests.post(url, json=data) print(response.json())

-

Run the client script to test the endpoint:

- `python test_ Résumé_parsing.py`

Database Setup with SQLAlchemy

This document provides instructions for setting up a database using SQLAlchemy in your Flask application.

Install SQLAlchemy and Other Dependencies

First, ensure SQLAlchemy and other required packages are installed in your virtual environment.

Install SQLAlchemy and Other Dependencies

1. Open your terminal or command prompt.
2. Run the following command to install SQLAlchemy and other dependencies:
   - `pip install flask-sqlalchemy flask-migrate flask-bcrypt flask-login`

Initialize SQLAlchemy in Flask

Initialize SQLAlchemy in your Flask application.

Update __init__.py in the app directory:

Initialize SQLAlchemy in Flask

- Add the following code to your `__init__.py`:

  `from flask import Flask from flask_sqlalchemy import SQLAlchemy from flask_migrate import Migrate from flask_bcrypt import Bcrypt from flask_login import LoginManager from config import Config`

```
db = SQLAlchemy()
migrate = Migrate()
bcrypt = Bcrypt()
login_manager = LoginManager()
login_manager.login_view = 'login'
login_manager.login_message_category = 'info'
```

```
def create_app(config_class=Config):
    app = Flask(__name__, template_folder='../templates')
    app.config.from_object(config_class)

    db.init_app(app)
    migrate.init_app(app, db)
    bcrypt.init_app(app)
    login_manager.init_app(app)

    from app import routes  # Import routes module
    routes.init_routes(app)  # Initialize routes

    from app.models import User  # Import User model for login
management

    @login_manager.user_loader
    def load_user(user_id):
        return User.query.get(int(user_id))

    return app
```

Create Database Models

Define your database models using SQLAlchemy. Create a `models.py` file in the `app` directory and add your models.

Example models.py:

Create Database Models

- Add the following code to your `models.py`:

```
from app import db
from flask_login import UserMixin

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(64), nullable=False)
```

```
    last_name = db.Column(db.String(64), nullable=False)
    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(128))
    created_at = db.Column(db.DateTime,
default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime,
default=db.func.current_timestamp(),
onupdate=db.func.current_timestamp())

    def __repr__(self):
        return f'<User {self.username}>'

class Resume(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    content = db.Column(db.Text)
    uploaded_at = db.Column(db.DateTime,
default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime,
default=db.func.current_timestamp(),
onupdate=db.func.current_timestamp())

class Feedback(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    resume_id = db.Column(db.Integer, db.ForeignKey('resume.id'))
    content = db.Column(db.Text)
    created_at = db.Column(db.DateTime,
default=db.func.current_timestamp())

class Template(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128))
    file_path = db.Column(db.String(256))
    created_at = db.Column(db.DateTime,
default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime,
default=db.func.current_timestamp(),
```

```
onupdate=db.func.current_timestamp())

class ResumeTemplate(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    resume_id = db.Column(db.Integer, db.ForeignKey('resume.id'))
    template_id = db.Column(db.Integer,
db.ForeignKey('template.id'))
    applied_at = db.Column(db.DateTime,
default=db.func.current_timestamp())

class Skill(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128))

class ResumeSkill(db.Model):
    resume_id = db.Column(db.Integer, db.ForeignKey('resume.id'),
primary_key=True)
    skill_id = db.Column(db.Integer, db.ForeignKey('skill.id'),
primary_key=True)

class JobDescription(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(128))
    description = db.Column(db.Text)
    created_at = db.Column(db.DateTime,
default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime,
default=db.func.current_timestamp(),
onupdate=db.func.current_timestamp())

class UserSession(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    session_token = db.Column(db.String(128), unique=True)
    created_at = db.Column(db.DateTime,
default=db.func.current_timestamp())
    expires_at = db.Column(db.DateTime)
```

```
class Log(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    action = db.Column(db.String(128))
    timestamp = db.Column(db.DateTime,
default=db.func.current_timestamp())

class Admin(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(128))
    created_at = db.Column(db.DateTime,
default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime,
default=db.func.current_timestamp(),
onupdate=db.func.current_timestamp())

class TemplateCategory(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128))
    description = db.Column(db.Text)

class TemplateCategoryMapping(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    template_id = db.Column(db.Integer,
db.ForeignKey('template.id'))
    category_id = db.Column(db.Integer,
db.ForeignKey('template_category.id'))

class FeedbackTemplate(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(128))
    content = db.Column(db.Text)
    created_at = db.Column(db.DateTime,
default=db.func.current_timestamp())
```

Configure the Database URI

Configure the database URI in the `config.py` file to point to your database.

Example config.py:

Configure Database URI

- Add or update the following code in your `config.py`:

```
import os
from sqlalchemy.engine import URL

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-
guess'
    database_url = URL.create(
        "mssql+pyodbc",
        username="",
        password="",
        host="BRIANS-DESKTOP\\SQLEXPRESS",
        database="ResuMate",
        query={"driver": "ODBC Driver 17 for SQL Server",
"trusted_connection": "yes"}
    )
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or
str(database_url)
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SQLALCHEMY_ECHO = True
```

Migrate the Database

Use Flask-Migrate to handle database migrations.

Install Flask-Migrate

Install Flask-Migrate

- Run the following command to install Flask-Migrate:

```
pip install flask-migrate
```

Initialize Flask-Migrate

Initialize Flask-Migrate

- Add the following code to your `__init__.py`:

```
from flask_migrate import Migrate

migrate = Migrate()

def create_app(config_class=Config):
    app = Flask(__name__, template_folder='../templates')
    app.config.from_object(config_class)

    db.init_app(app)
    migrate.init_app(app, db)
    bcrypt.init_app(app)
    login_manager.init_app(app)

    from app import routes  # Import routes module
    routes.init_routes(app)  # Initialize routes

    from app.models import User  # Import User model for login
management

    @login_manager.user_loader
    def load_user(user_id):
        return User.query.get(int(user_id))

    return app
```

Create and Apply Migrations

Create and Apply Migrations

- Run the following commands to create and apply database migrations:
  - `flask db init` (only run this once to initialize migrations)
  - `flask db migrate -m "Initial migration"`
  - `flask db upgrade`

Conclusion

Setting up SQLAlchemy with Flask allows for a powerful and flexible database solution for your application. By following these steps, you can ensure a properly configured database setup and manage migrations effectively.

Routes and Views

This document provides instructions for setting up routes and views in your Flask application.

Define Routes

Create routes in the `routes.py` file to handle different web requests.

Example routes.py:

Define Routes

- Add the following code to your `routes.py`:

```
from flask import render_template, request, jsonify, current_app as
app
from .nlp import parse_resume

@app.route('/')
def index():
return render_template('index.html')

@app.route('/upload', methods=['GET', 'POST'])
def upload_resume():
if request.method == 'POST':
resume_text = request.files['resume'].read().decode('utf-8')
parsed_resume = parse_resume(resume_text)
return jsonify(parsed_resume)
return render_template('upload.html')

@app.route('/parse_resume', methods=['POST'])
def parse_resume_route():
resume_text = request.json.get('resume_text', '')
parsed_resume = parse_resume(resume_text)
return jsonify(parsed_resume)
```

Create HTML Templates

Create HTML templates for rendering web pages. These templates should be placed in the `templates` directory.

Example index.html:

Create index.html

- Create a new file `index.html` in the `templates` directory and add the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>ResuMate</title>
</head>
<body>
    <h1>Welcome to ResuMate</h1>
        <a href="/upload">Upload Resume</a>
    </body>
</html>
```

Example upload.html:

Create upload.html

- Create a new file `upload.html` in the `templates` directory and add the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
<title>Upload Resume</title>
</head>
<body>
    <h1>Upload Your Resume</h1>
        <form action="/upload" method="post"
enctype="multipart/form-data">
            <input type="file" name="resume">
            <input type="submit" value="Upload">
        </form>
</body>
</html>
```

Test Routes and Views

Ensure that your routes and views are working correctly by running the Flask application and visiting the corresponding URLs.

Test Routes and Views

1. Start the Flask application by running the following command in your terminal or command prompt:

   ```
   python run.py
   ```

2. Visit the following URLs in your web browser to test the routes and views:

   - `http://127.0.0.1:5000/` - To view the index page.
   - `http://127.0.0.1:5000/upload` - To view the upload resume page.

Conclusion

Setting up routes and views in Flask involves defining routes in the `routes.py` file and creating HTML templates for rendering web pages. By following these steps, you can ensure a properly configured and functional web application.

Test Cases

This document outlines the test cases designed to verify the functionality of various components in the ResuMate project.

Overview

Each test case is designed to validate a specific functionality or feature within the application. The test cases are organized by module and cover a range of scenarios, including normal operation, edge cases, and potential failure conditions.

Test Case Structure

Each test case includes the following information:

- **Test Case ID**: A unique identifier for the test case.
- **Description**: A brief summary of the test case.
- **Preconditions**: Any setup or conditions that must be met before executing the test.
- **Test Steps**: Detailed steps to execute the test.
- **Expected Results**: The expected outcome of the test.
- **Actual Results**: The actual outcome of the test (to be filled after execution).

User Authentication

Test Case 1: User Registration

- **Test Case ID**: TC-USER-001
- **Description**: Verify that a new user can successfully register.
- **Preconditions**: None
- **Test Steps**:

    1. Navigate to the registration page.

2. Enter valid user details (username, email, password).
3. Submit the registration form.

- **Expected Results**: The user is successfully registered and redirected to the login page.

Test Case 2: User Login

- **Test Case ID**: TC-USER-002
- **Description**: Verify that a registered user can log in.
- **Preconditions**: User must be registered.
- **Test Steps**:
  1. Navigate to the login page.
  2. Enter valid credentials (username, password).
  3. Submit the login form.
- **Expected Results**: The user is successfully logged in and redirected to the dashboard.

File Upload

Test Case 1: Resume Upload

- **Test Case ID**: TC-UPLOAD-001
- **Description**: Verify that a user can upload a resume file.
- **Preconditions**: User must be logged in.
- **Test Steps**:
  1. Navigate to the upload page.
  2. Select a valid resume file.
  3. Submit the upload form.

- **Expected Results**: The resume file is successfully uploaded, and a confirmation message is displayed.

Test Case 2: Invalid File Upload

- **Test Case ID**: TC-UPLOAD-002
- **Description**: Verify that an invalid file type cannot be uploaded.
- **Preconditions**: User must be logged in.
- **Test Steps**:
  1. Navigate to the upload page.
  2. Select an invalid file type (e.g., .exe, .png).
  3. Submit the upload form.
- **Expected Results**: The file is not uploaded, and an error message is displayed.

Resume Parsing

Test Case 1: Parse Valid Resume

- **Test Case ID**: TC-PARSE-001
- **Description**: Verify that a valid resume file is parsed correctly.
- **Preconditions**: A valid resume file must be uploaded.
- **Test Steps**:
  1. Upload a valid resume file.
  2. Initiate the parsing process.
- **Expected Results**: The resume file is parsed successfully, and the extracted information is displayed.

Test Case 2: Parse Invalid Resume

- **Test Case ID**: TC-PARSE-002
- **Description**: Verify that an invalid resume file does not cause the application to crash.
- **Preconditions**: An invalid resume file must be uploaded.
- **Test Steps**:
  1. Upload an invalid resume file.
  2. Initiate the parsing process.
- **Expected Results**: The application handles the error gracefully, and an error message is displayed.

Conclusion

These test cases ensure that key functionalities within the ResuMate project are working correctly. Regular execution of these tests helps maintain the stability and reliability of the application. Additional test cases should be created as new features are developed.

Upcoming Features

This document provides an overview of the upcoming features planned for the ResuMate project. These features aim to enhance the platform's functionality, user experience, and overall value to jobseekers and employers.

Advanced Résumé Analysis

Description

Enhance the Résumé analysis capabilities by incorporating more sophisticated AI and ML techniques.

Features

- **Keyword Matching**: Match Résumé content with job descriptions to identify relevant keywords.
- **Sentiment Analysis**: Analyze the tone and sentiment of the Résumé to ensure a positive impression.
- **Contextual Analysis**: Evaluate the context and coherence of the Résumé content.

Benefits

- Provides deeper insights into Résumé quality.
- Helps users tailor their Résumés to specific job descriptions.
- Ensures that the Résumé conveys a professional and positive tone.

Real-Time Feedback

Description

Offer real-time feedback as users edit their Résumés, providing instant suggestions and improvements.

Features

- **Live Editing**: Show suggestions and corrections as users type.
- **Interactive Tips**: Provide tips and best practices for each section of the Résumé.
- **Error Highlighting**: Highlight grammar, spelling, and formatting errors in real-time.

Benefits

- Improves the quality of Résumés during the creation process.
- Saves users time by providing instant feedback.
- Enhances the overall user experience with interactive guidance.

Collaboration Tools

Description

Introduce features that allow users to collaborate with peers, mentors, or career coaches on their Résumés.

Features

- **Shared Editing**: Enable multiple users to edit a Résumé simultaneously.
- **Commenting System**: Allow collaborators to leave comments and suggestions.
- **Version Control**: Track changes and maintain version history of the Résumé.

Benefits

- Facilitates peer reviews and mentor guidance.
- Enhances the quality of Résumés through collaborative efforts.
- Provides a structured approach to Résumé editing and feedback.

Integration with Job Portals

Description

Integrate ResuMate with popular job portals to streamline the job application process.

Features

- **Direct Application**: Allow users to apply for jobs directly from the platform.
- **Profile Syncing**: Sync Résumé data with profiles on job portals.
- **Job Recommendations**: Provide personalized job recommendations based on Résumé content.

Benefits

- Simplifies the job application process.
- Ensures consistency between Résumés and job portal profiles.
- Helps users discover relevant job opportunities.

Mobile Application

Description

Develop a mobile application for ResuMate, providing users with on-the-go access to the platform.

Features

- ** Résumé Editing**: Enable users to create and edit Résumés from their mobile devices.
- **Push Notifications**: Send notifications for feedback, updates, and job recommendations.
- **Offline Access**: Allow users to access and edit Résumés without an internet connection.

Benefits

- Increases accessibility and convenience for users.
- Keeps users engaged with timely notifications and updates.

- Provides flexibility with offline editing capabilities.

Advanced Analytics and Insights

Description

Introduce advanced analytics and insights to help users understand and improve their Résumé performance.

Features

- ** Résumé Performance Metrics**: Track views, downloads, and feedback on Résumés.
- **Comparative Analysis**: Compare Résumé performance against industry benchmarks.
- **Actionable Insights**: Provide specific recommendations based on analytics.

Benefits

- Empowers users with data-driven insights.
- Helps users continuously improve their Résumés.
- Provides a competitive edge in the job market.

Customizable Templates

Description

Expand the library of Résumé templates and allow users to fully customize them to suit their preferences.

Features

- **Template Library**: Offers a wide range of professional templates.
- **Customization Options**: Allow users to customize fonts, colors, layouts, and sections.
- **Template Preview**: Provide real-time previews of template changes.

Benefits

- Enhances the visual appeal of Résumés.
- Provides flexibility for users to create unique and personalized Résumés.
- Improves user satisfaction with diverse template options.

Conclusion

The upcoming features for the ResuMate project are designed to enhance the platform's functionality, user experience, and overall value. By implementing these features, ResuMate aims to provide a comprehensive and user-centric solution for jobseekers, helping them create effective and professional Résumés that stand out in the competitive job market.

Future Sprints

This document outlines the plan for future sprints in the ResuMate project. Each sprint focuses on specific tasks and milestones to ensure continuous development and improvement of the platform.

Sprint 5: Machine Learning Integration

Objectives

- Train NLP models for resume analysis.
- Develop a user authentication system.
- Refine feedback generation based on ML insights.

Tasks

- Collect and preprocess additional resume data for training.
- Train and validate NLP models using TensorFlow or PyTorch.
- Implement user authentication using Flask-Login.
- Integrate the trained NLP models with the backend.
- Enhance the feedback system with machine learning insights.

Deliverables

- Trained NLP models.
- User authentication system.
- Improved feedback generation logic.

Sprint 6: Frontend Enhancements

Objectives

- Design and implement advanced UI components.
- Improve the user interface for better usability and aesthetics.
- Ensure mobile responsiveness and accessibility.

Tasks

- Develop interactive UI components using Vue.js.
- Enhance existing templates with CSS improvements.
- Conduct user testing to gather feedback on UI/UX.
- Implement accessibility features according to WCAG guidelines.

Deliverables

- Advanced and interactive UI components.
- Improved and accessible user interface.

Sprint 7: Data Analytics and Reporting

Objectives

- Implement data analytics features to provide insights to users.
- Develop reporting tools for user activity and resume performance.
- Integrate analytics with the backend and frontend.

Tasks

- Set up data analytics tools and frameworks (e.g., Google Analytics).
- Develop backend endpoints for data collection and processing.
- Create frontend components to display analytics and reports.
- Test and validate the accuracy of analytics and reports.

Deliverables

- Data analytics and reporting features.
- User activity and performance reports.

Sprint 8: Template Expansion

Objectives

- Expand the library of resume and CV templates.
- Implement industry-specific templates to cater to various job sectors.
- Allow users to customize and save their templates.

Tasks

- Design new templates for different industries (e.g., healthcare, finance).
- Add template customization options (e.g., colors, fonts, layouts).
- Implement a template management system in the backend.
- Conduct user testing to gather feedback on new templates.

Deliverables

- Expanded template library.
- Customizable and industry-specific templates.

Sprint 9: Security and Compliance

Objectives

- Enhance security measures to protect user data.
- Ensure compliance with GDPR, CCPA, and other relevant regulations.

- Implement data encryption and secure storage solutions.

Tasks

- Conduct a security audit to identify potential vulnerabilities.
- Implement data encryption for sensitive information.
- Update privacy policy and terms of service to ensure compliance.
- Develop tools for user data management (e.g., data export, deletion).

Deliverables

- Enhanced security measures.
- Compliance with data protection regulations.
- Updated privacy policy and terms of service.

Sprint 10: User Feedback and Iteration

Objectives

- Gather and analyze user feedback to identify areas for improvement.
- Implement changes based on user feedback to enhance the platform.
- Plan future features and updates based on user needs.

Tasks

- Develop surveys and feedback forms to collect user opinions.
- Analyze feedback to identify common issues and suggestions.
- Prioritize and implement changes based on feedback.
- Plan the next set of features and updates for future sprints.

Deliverables

- User feedback analysis.
- Platform improvements based on feedback.
- Roadmap for future features and updates.

Conclusion

The future sprints for the ResuMate project are designed to ensure continuous development, improvement, and expansion of the platform. By following this plan, the ResuMate team aims to deliver a robust, user-centric application that meets the evolving needs of job seekers and employers.

Sprint Retrospective

This document provides a reflection on the progress made during the current sprint, including successes, challenges, and areas for improvement.

What Went Well

- **Environment Setup**: Successfully set up the development environment and resolved initial issues with virtual environment activation.
- **Project Structure**: Established a clear and organized project structure, making it easy to navigate and manage.
- **Implementation**: Implemented key components such as SQLAlchemy for database management, SpaCy for resume parsing, and Flask for the web application framework.
- **Documentation**: Created comprehensive documentation for each phase of the project, ensuring that all steps are well-documented and easy to follow.
- **Testing**: Set up the testing environment and created initial test cases, ensuring that the codebase is robust and reliable.

What Didn't Go Well

- **Time Management**: Encountered delays due to initial setup issues and miscommunication about the documentation format.
- **Tool Familiarity**: Faced challenges with using Writerside for documentation, resulting in some inefficiencies and confusion.
- **Dependency Issues**: Experienced difficulties with dependency installation and management, particularly with ensuring all team members had the same environment setup.

What We Can Improve

- **Clear Communication**: Improve communication about tool usage and documentation standards to avoid misunderstandings and ensure consistency.
- **Time Estimation**: Better estimate time required for tasks to avoid delays and ensure timely completion of sprint goals.
- **Tool Training**: Provide training sessions or resources on using Writerside and other tools to improve efficiency and reduce setup time.

Action Items for Next Sprint

- **Enhanced Documentation**: Continue improving the documentation process, ensuring all steps are clearly outlined and easy to follow.
- **Environment Standardization**: Create a standardized setup script or guide to ensure all team members have the same development environment.
- **Regular Check-ins**: Implement regular check-ins to monitor progress, address issues promptly, and ensure alignment on sprint goals.

Conclusion

The sprint was productive and resulted in significant progress, despite some challenges. By addressing the identified areas for improvement and implementing the action items, we can ensure a smoother and more efficient workflow in the next sprint.

Completed Tasks

This document provides an overview of the tasks that have been completed for the ResuMate project.

Project Setup

- **Environment Setup**: Successfully set up the development environment with Python and a virtual environment.
- **Project Structure**: Created the initial project structure with appropriate directories and files.

Implementation Details

- **Database Setup with SQLAlchemy**: Implemented SQLAlchemy for database management in the Flask application.
- **Flask Application Setup**: Set up the basic Flask application structure, including `__init__.py`, `routes.py`, and configuration.
- **Resume Parsing with SpaCy**: Integrated SpaCy for natural language processing to parse resume content.
- **Routes and Views**: Developed the initial routes and views for the Flask application.

Introduction

- **Objectives**: Defined the objectives of the ResuMate project.
- **Project Overview**: Provided a comprehensive overview of the ResuMate project.
- **Technologies Used**: Listed and explained the technologies used in the project.

Next Steps

- **Future Sprints**: Outlined the tasks and goals for future sprints.
- **Upcoming Features**: Identified and described the features to be developed in the upcoming phases.

Testing

- **Running Tests**: Set up the testing environment and created initial test cases for the application.

Sprint Review and Retrospective

- **Completed Tasks**: Documented the tasks completed so far in this file.
- **Sprint Retrospective**: Reflected on the progress and identified areas for improvement in the project workflow.

Running Tests

This document provides instructions on how to run tests for the ResuMate project to ensure that all components are functioning correctly.

Prerequisites

Before running tests, ensure you have the following prerequisites set up:

- Python is installed and the virtual environment is activated.
- All project dependencies are installed as outlined in the Dependencies Installation (Dependencies Installation) document.
- The testing framework (unittest) is installed.

Running Tests

Run Tests

1. Activate the virtual environment if it is not already activated:
   - **Windows:**
     - `.\.venv\Scripts\Activate.ps1`
   - **macOS/Linux:**
     - `source .venv/bin/activate`
2. Navigate to the root directory of your project:

   `cd path/to/ResuMate`
3. Run the tests using the unittest framework with the following command:

   `python -m unittest discover -s test -p "*.py"`

Viewing Test Results

After running the tests, you will see the results displayed in the terminal or command prompt. The results will indicate the number of tests that passed, failed, or were skipped.

Debugging Test Failures

If any tests fail, follow these steps to debug and resolve the issues:

Debug Test Failures

1. Identify the test that failed and read the error message provided in the test output.
2. Open the corresponding test file in your code editor and review the test case.
3. Check the code being tested for potential issues. Ensure that all dependencies and configurations are correctly set up.
4. Fix any identified issues in the code or test case.
5. Re-run the tests to ensure the issue is resolved:

   `python -m unittest discover -s test -p "*.py"`

Best Practices for Writing Tests

To ensure comprehensive and maintainable tests, follow these best practices:

- **Isolate Tests**: Ensure each test is independent and does not rely on the state or results of other tests.
- **Use Meaningful Test Names**: Name your tests descriptively to indicate what functionality they are verifying.
- **Test Edge Cases**: Include tests for edge cases and potential failure scenarios to ensure robustness.
- **Keep Tests Simple**: Write clear and concise tests to make them easy to understand and maintain.
- **Automate Testing**: Integrate tests into your continuous integration (CI) pipeline to automate test execution on each code commit.

Conclusion

Running tests is essential for maintaining the stability and reliability of the ResuMate project. By following these instructions and best practices, you can ensure that all components are thoroughly tested and any issues are promptly identified and resolved.