A Technical Report on the ‘cv_analysis_backend’ component of Play2Earn.ai

15th September 2024

Robin Rai

Contents

Overview

The concept of Play2Earn.ai revolves around users playing games enhanced by AI. By contributing to learning models while playing, users earn cryptocurrency as a reward. For this report, three key components divide the current iteration of development:

- **Backend** (Game and cryptocurrency management)
- **CV Processing and User Data Backend** (Handles user authentication, CV uploads, and analysis
- **Frontend** (Built using React, for user interaction)

This report will focus on the **CV Processing and User Data Backend** component of Play2Earn.ai, where users can upload resumes, have them analysed using machine learning models, and get the data extracted stored securely. This component also handles user authentication using **JSON Web Tokens (JWT)** and stores its data in **MongoDB**.

Key **aspects** of this component include:

1. **User authentication and session management**
2. **CV upload and processing**, including parsing, virus scanning, and similarity analysis (to a given job description).
3. **Machine learning-based resume analysis**.
4. **Database storage and management** of user data and analysed CVs.

The report will delve into the system architecture and technical features I have identified and explored during my research, along with suggestions for potential improvements.

System Architecture and Technical Features

1. User Authentication and Session Management

The system uses **JWT** to authenticate users. After reviewing the source code, I found that the authentication system is robust and secure. Also, as **JWT** is for stateless authentication, no user session data is stored on the server which is great for scalability.

The process works as follows:

- **Token Generation:** Upon logging in, they are issued a **JWT** token. This token is then used for all subsequent requests to secure endpoints such as CV upload and analysis. This ensures that the user's identity can be verified without the need for session storage.

- **Token Validation:** The validity of the token is checked before committing any of these sensitive actions. If expired or invalid, access is denied. This is handled with a utility function in '**app.py**':

```
37  # Manual JWT verification
38  def verify_jwt(token):
39      try:
40          decoded = decode(token, app.config['JWT_SECRET_KEY'], algorithms=[app.config['JWT_ALGORITHM']])
41          return decoded
42      except ExpiredSignatureError:
43          return {"msg": "Token has expired"}
44      except InvalidTokenError:
45          return {"msg": "Invalid token"}
```

The token is passed in the **Authorization header**, formatted as a Bearer token. The system also employs **Cross-Origin Resource Sharing (CORS)** which allows communication between the frontend and backend.

In conclusion, the system's use of **JWT** makes it great for scalability and secure for user authentication. It is a great design choice for the goal of handling large numbers of users, especially as it does not store the session data directly onto the server.

2. CV Upload and Processing

A core feature of the component is the ability to upload and analyse CVs. During my research, I explored the mechanism to upload files along with the subsequent steps which handle file validation, virus scanning, and text extraction.

A basic outline of the process is as follows:

- **File validation:** The system checks the file extension of the uploaded file to ensures it's of valid type (.pdf, .docx, .doc). This is handled by the '**allowed_file()**' function, ensuring only supported file types (the ones listed in '**allowed_extensions**') are processed.

```python
class Config:
    SECRET_KEY = os.getenv('SECRET_KEY')
    JWT_SECRET_KEY = os.getenv('JWT_SECRET_KEY')
    UPLOAD_FOLDER = 'uploads'
    ALLOWED_EXTENSIONS = {'pdf', 'docx', 'doc'}

def allowed_file(filename, allowed_extensions):
    """Check if the uploaded file is allowed based on its extension."""
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in allowed_extensions
```

- **Virus Scanning:** After validation, a virus scan using the **Cloudmersive Virus Scan API** is used. This ensures that the uploaded file is secure before processing it any further.
- **Text Extraction:** Only once the file is cleared by the virus scan the contents are extracted. This extraction process is handled differently based on the format of the file:
    - For PDF files: **PyMuPDF** library is used to extract text from the pages.
    - For DOCX files: the **docx** library is used to read the content of the document.
    - For DOC files: like DOCX files, but with a fallback method in case it fails to read correctly.

I can say this process is well-structured, however, areas such as error handling could be improved. For instance, when extraction fails, an error is logged by the system, but no information is provided to the user of what went wrong. I understand that this component has only recently been established and believe providing more granular feedback for different error types (e.g., unsupported format or corrupted file) would improve the user experience.

3. Machine Learning-Based Resume Analysis

One of the more advanced and fundamentally exciting features I researched is the resume analysis system, which uses machine learning models to compare the uploaded CV against a given job description. With inspection of the source code, I can see that the system leverages scikit-learn to compute the **TF-IDF (Term Frequency-Inverse Document Frequency)** scores and calculate **cosine similarity** between the CV and job description.

- **Text Preprocessing:** Both the CV and job description have their contents converted to lowercase for consistency. In the source code, additional preprocessing techniques are proposed such as **lemmatisation** and **stopwords** removal but have yet to be implemented. This stage prepares the text for vectorisation.
- **TF-IDF Vectorisation:** The system tokenises the text and uses **TF-IDF** to represent the importance of words in both the CV and job description. How this works is essentially by ensuring common words (such as ‘and’ or ‘the’) are weighted less heavily than more rarer and key terms like ‘data analysis’.
- **Cosine Similarity:** The system then computes the **cosine similarity** score, a metric which measures the cosine of the angle between two vectors (the two vectorised texts from CV and the job description in this context), providing a normalised measure of similarity. Here the score reflects how closely the CV matches the job description in terms of content. The higher the **cosine similarity**, the more relevant the CV is to the job description.

The resulting context score is stored to the database and is accessible to users when they check their CV analysis results. I found this feature to be a well-implemented and useful tool with multiple practical applications for it (tailoring CVs to job applications or ranking most suited candidate for a job based on these scores).

4. Database Storage and Management

For storing user data and analysed CVs, the system uses **MongoDB** with models defined in **MongoEngine**. The data models seem to be well-structured, with two key collections:

- **'User'** Collection: Stores the username and other personal information for registered users. This allows a connection between the user and their uploaded CVs.
- **'CVAnalysis'** Collection: Stores the details of the analysed CVs including a multitude of features.

```python
class CVAnalysis(me.Document):
    file_name = me.StringField(max_length=255, null=True)
    context_score = me.FloatField(null=True)
    user = me.ReferenceField(User, required=True, reverse_delete_rule=me.CASCADE)
    name = me.StringField(max_length=255, null=True)
    designation = me.StringField(max_length=255, null=True)
    experience = me.StringField(max_length=255, null=True)
    education = me.StringField(max_length=255, null=True)
    skills = me.StringField(max_length=255, null=True)
    salary_expectation = me.StringField(max_length=255, null=True)
    description = me.StringField(null=True)
    professional_title = me.StringField(max_length=255, null=True)
    years_of_experience = me.StringField(max_length=255, null=True)

    meta = {
        'collection': 'cv_analysis',
        'db_alias': 'cvanalysis_db'
    }
```

I noted that **MongoDB** was chosen over a relational database (**MySQL**), which was initially used in the early stages of the project (suggested from the existence of '**models_mysql_deprecated.py**'). The transition to **MongoDB** seems just and a good fit for this component due to the nature of complex data like CVs which **MongoDB** accommodates with flexible, document-based storage.

Potential Improvements

After conducting my research and analysis of this component, I have identified a few areas for potential improvement that could enhance the scalability, efficiency, and user experience:

1. *Asynchronous Task Processing*

Currently, CV uploads and processing are handled synchronously. The process does features parallel processing which implements the use of other CPU cores which is great for speed and extensive tasks presented here.

```
# Processing CVs with multiprocessing
def process_cvs_in_chunks(cv_files, cv_folder_path, chunk_size=100):
    results = []
    for i in range(0, len(cv_files), chunk_size):
        chunk_files = cv_files[i:i + chunk_size]
        file_paths = [os.path.abspath(os.path.join(cv_folder_path, f)) for f in chunk_files]
        with Pool(cpu_count()) as p:
            chunk_results = p.map(process_resume, file_paths)
        results.extend(chunk_results)
    return results
```

However, introducing **asynchronous task processing** would enable the various steps to run in the background without blocking other requests. This would allow users to receive immediate feedback while also ensuring the server doesn't get bogged down if it was to handle a multitude of users at the same time.

2. *Enhanced Skill and Experience Matching*

The system currently uses keyword-based extraction for skills and experience (as showcased in the function **'extract_skills()'** in **'utils.py'**). Perhaps by implementing **Named Entity Recognition (NER)** models it could allow for more accurate extraction of these key entities and therefore better context scores. The way this would work is that nuanced, or less common terms associated with the keywords would be recognised by the system even if they're not directly predefined in the list.

```
def extract_skills(text):
    """Extract the skills from the CV text."""
    skill_keywords = ['Python', 'Java', 'C++', 'Project Management', 'Data Analysis', 'SQL',
    found_skills = []
    for keyword in skill_keywords:
        # Use re.escape() to handle special characters in skill names
        if re.search(r'\b{}\b'.format(re.escape(keyword)), text, re.IGNORECASE):
            found_skills.append(keyword)
    return ', '.join(found_skills) if found_skills else None
```

3. *Improved Error Handling and User Feedback*

I do completely understand this component is still in early iterations of development. Therefore, as development progresses it would make sense to have improved error handling with more meaningful feedback provided to the user by the system.

Conclusion

Ultimately, this component seems to be a well-architected system. Machine learning, Natural Language Processing (NLP), and authentication technologies build up a robust platform for the tasks desired by the component. **MongoDB** integration allows for flexible data storage and **JWT** for authentication makes the system scalable and secure.

There are, however, room for improvement with features such as asynchronous processing or better error handling. By addressing these issues, it would further enhance the performance and user experience of this component leading to overall greater scalability and efficiency of Play2Earn.ai.

This report reflects my research and understanding of the **CV Processing and User Data Backend** component of Play2Earn.ai. I'm excited about the potential of Play2Earn.ai and understand this component plays a critical role in delivering the seamless, AI-powered user experience which greatly benefits the platform.

END OF REPORT