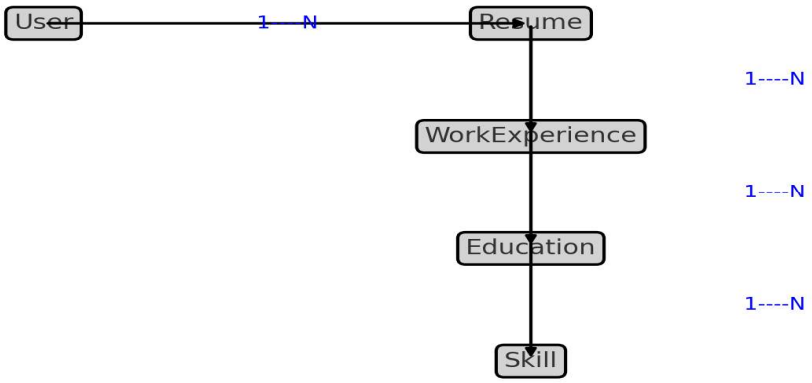
ONLINE RESUME BUILDER USING DJANGO

ER DIAGRAM:



The ER diagram represents the database structure for the Online Resume Builder application. It shows the entities involved and the relationships between them. Here's a detailed explanation of each component:

Entities and Relationships

1. **User**
   - The User entity represents the users of the application. Each user can create multiple resumes.
   - Attributes:
     - id: Primary Key (PK), unique identifier for each user.
     - username: The username of the user.
     - password: The password for the user account.
     - email: The email address of the user.
2. **Resume**
   - The Resume entity represents the resume created by the user. Each user can have multiple resumes, and each resume belongs to a single user.
   - Relationships:
     - One-to-Many (1) relationship with the User entity: One user can have many resumes.
   - Attributes:
     - id: Primary Key (PK), unique identifier for each resume.
     - user_id: Foreign Key (FK) linking to the User entity.
     - title: The title of the resume.
     - summary: A brief summary or objective statement in the resume.

3. **WorkExperience**
    - The WorkExperience entity represents the work experiences listed in a resume. Each resume can have multiple work experiences, and each work experience belongs to a single resume.
    - Relationships:
        - One-to-Many (1

) relationship with the Resume entity: One resume can have many work experiences.

    - Attributes:
        - id: Primary Key (PK), unique identifier for each work experience.
        - resume_id: Foreign Key (FK) linking to the Resume entity.
        - job_title: The job title of the work experience.
        - company: The company where the work experience was gained.
        - start_date: The start date of the work experience.
        - end_date: The end date of the work experience.
4. **Education**
    - The Education entity represents the educational qualifications listed in a resume. Each resume can have multiple educational qualifications, and each education record belongs to a single resume.
    - Relationships:
        - One-to-Many (1

) relationship with the Resume entity: One resume can have many education records.

    - Attributes:
        - id: Primary Key (PK), unique identifier for each education record.
        - resume_id: Foreign Key (FK) linking to the Resume entity.
        - degree: The degree or qualification obtained.
        - institution: The institution where the education was obtained.
        - start_date: The start date of the education.
        - end_date: The end date of the education.
5. **Skill**
    - The Skill entity represents the skills listed in a resume. Each resume can have multiple skills, and each skill belongs to a single resume.
    - Relationships:
        - One-to-Many (1

) relationship with the Resume entity: One resume can have many skills.

    - Attributes:
        - id: Primary Key (PK), unique identifier for each skill.
        - resume_id: Foreign Key (FK) linking to the Resume entity.
        - name: The name of the skill.

DB CODE:

```
from django.db import models

from django.contrib.auth.models import User

class Resume(models.Model):

    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='resumes')

    title = models.CharField(max_length=100)

    summary = models.TextField()

    def __str__(self):

        return self.title

class WorkExperience(models.Model):

    resume = models.ForeignKey(Resume, on_delete=models.CASCADE,
related_name='work_experiences')

    job_title = models.CharField(max_length=100)

    company = models.CharField(max_length=100)

    start_date = models.DateField()

    end_date = models.DateField()

    def __str__(self):

        return f'{self.job_title} at {self.company}'

class Education(models.Model):

    resume = models.ForeignKey(Resume, on_delete=models.CASCADE,
related_name='educations')

    degree = models.CharField(max_length=100)

    institution = models.CharField(max_length=100)

    start_date = models.DateField()

    end_date = models.DateField()

    def __str__(self):
```

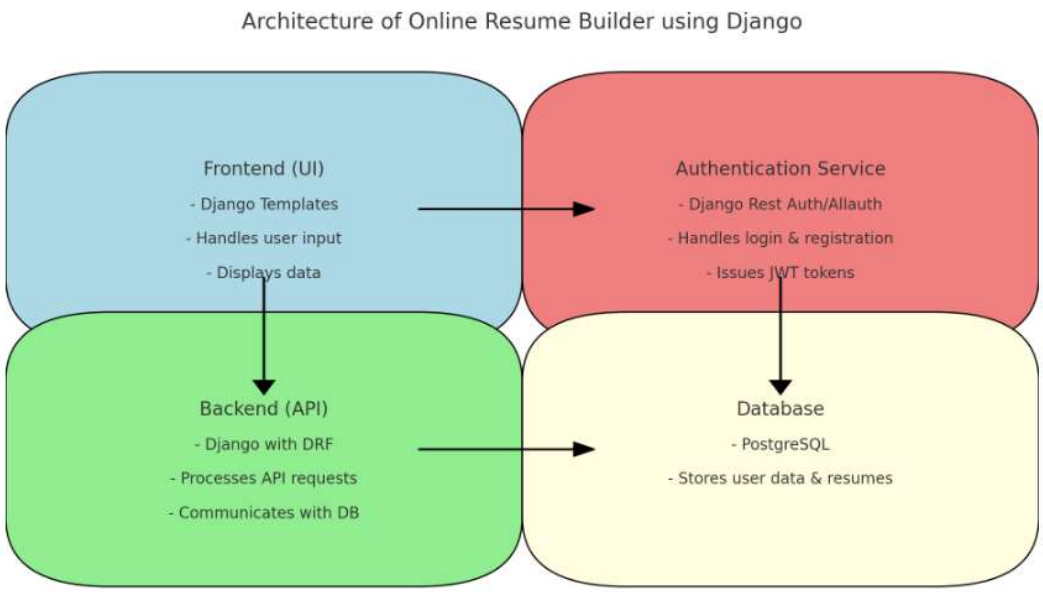
```
        return f'{self.degree} from {self.institution}'

class Skill(models.Model):

    resume = models.ForeignKey(Resume, on_delete=models.CASCADE, related_name='skills')

    name = models.CharField(max_length=100)

    def __str__(self):

        return self.name
```

1. Defining Models

In Django, you define your database schema using models in the models.py file. Each model class corresponds to a database table. Each attribute of the model represents a column in the table. Here's a breakdown of the key components:

- **Class Definition**: Each model class inherits from models.Model. This tells Django that the class is a model and should be mapped to a database table.
- **Fields**: Each attribute of the model class represents a column in the database table. The type of field (e.g., CharField, DateField, ForeignKey) determines the type of data stored in that column and how it is managed.
- **Relationships**: Django provides various types of relationships between models:
    - **ForeignKey**: Defines a many-to-one relationship. For example, each WorkExperience is linked to one Resume.
    - **ManyToManyField**: Defines a many-to-many relationship.
    - **OneToOneField**: Defines a one-to-one relationship.

ARCHITECTURE DIAGRAM:



The architecture of the Online Resume Builder application is designed to ensure a modular, scalable, and efficient system. At the core of the application is the **Front-End**, which provides the user interface (UI) and client-side logic. The UI consists of web pages and forms that users interact with to input their personal information, work experiences, education, and skills. Client-side logic, often implemented using JavaScript frameworks, manages dynamic content updates and form validations.

The **Back-End** of the application is built using Django, which handles the main application logic. This includes Views, which process incoming requests and interact with Django Models to fetch or manipulate data. The URLs component defines the endpoints that map to these views, ensuring that requests are routed correctly. Models define the structure of the data, while Serializers convert complex data types into JSON, facilitating communication between the front-end and back-end.

Data is stored in the **Database**, which can be PostgreSQL, MySQL, or SQLite. The database schema is defined by Django models, translating into tables, columns, and relationships within the database. This relational database manages user data, resumes, work experiences, education, and skills.

External services play a crucial role in augmenting the application's capabilities. This includes an **Email Service** for sending notifications and confirmations, and **Authentication Providers** if third-party authentication services are used.

The **API** layer is pivotal in connecting the front-end and back-end. It exposes RESTful endpoints that the front-end uses to perform CRUD (Create, Read, Update, Delete) operations on resources such as resumes and work experiences. The API ensures structured data exchange between the client and server, enabling seamless interaction.

In summary, the architecture comprises a well-defined separation of concerns: the front-end handles user interaction, the back-end manages business logic and data processing, the database stores application data, external services provide additional functionalities, and the API facilitates communication between the front-end and back-end. This design ensures a robust and maintainable application structure.