1. **Client-Server Architecture**:
    - An ATS usually follows a client-server architecture where client devices (browsers or mobile apps) interact with a central server.
    - The client interface is responsible for presenting the user interface, collecting user inputs, and displaying data.
2. **Web-Based Frontend**:
    - The frontend is typically implemented as a web application to ensure cross-platform accessibility.
    - It uses technologies like HTML, CSS, and JavaScript to create the user interface.
    - Responsive design ensures usability on different devices, including desktops, tablets, and mobile phones.
3. **Backend Server**:
    - The backend server is the core of the system and is responsible for processing user requests, managing data, and executing business logic.
    - It can be built using various technologies, such as:
        - **Programming Language**: Common choices include Python, Ruby, Node.js, Java, or C#.
        - **Web Framework**: Frameworks like Django, Ruby on Rails, Express.js, or ASP.NET can simplify development.
        - **Database Management System (DBMS)**: Use a relational database system like PostgreSQL, MySQL, or a NoSQL database like MongoDB to store and manage data.
        - **APIs**: Develop RESTful APIs to facilitate communication between the frontend and backend.
4. **Database Layer**:
    - The database layer is responsible for storing and retrieving data related to job listings, applicants, user accounts, and other system entities.
    - Use an appropriate database schema to efficiently organize and manage data.
    - Implement data access layers to interact with the database, following best practices for data security and privacy.
5. **Authentication and Authorization**:
    - Implement robust authentication and authorization mechanisms to ensure secure access control.
    - Technologies like OAuth, JWT, or OpenID Connect may be used for user authentication and authorization.
6. **Application Logic**:
    - Define the application's core logic, including workflows for job posting, application processing, and reporting.
    - Implement business rules and automation for tasks such as resume parsing, application status tracking, and interview scheduling.

7. **External Integrations**:
    - Integrate with external systems and services, such as job boards, social media platforms, email services, and HR tools.
    - Use APIs, webhooks, and other integration methods to exchange data and information seamlessly.
8. **Scalability and Load Balancing**:
    - Plan for scalability by using load balancing techniques and considering cloud-based hosting solutions.
    - Horizontal scaling allows the system to handle increasing traffic and data volumes.
9. **Security Measures**:
    - Implement security features to protect sensitive candidate data, including encryption, input validation, and user access controls.
    - Regularly update and patch system components to protect against vulnerabilities.
10. **Monitoring and Logging**:
    - Set up monitoring tools to track system performance, identify issues, and provide insights for optimization.
    - Create comprehensive logging to record user activities and system events for troubleshooting and auditing.
11. **Deployment and Hosting**:
    - Select a suitable hosting environment, whether it's on-premises servers, cloud platforms like AWS, Azure, or Google Cloud, or a combination of both.
    - Develop a deployment strategy that includes staging, testing, and production environments.
12. **Backup and Recovery**:
    - Implement data backup and recovery mechanisms to ensure data availability in case of system failures or data loss.
13. **Documentation and Knowledge Sharing**:
    - Create technical documentation that describes the architecture, system components, and how to maintain and troubleshoot the system.
    - Ensure knowledge sharing among the development and operations teams.