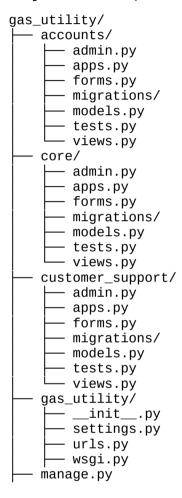
KONDRAKUNTA LAVAKUMAR Submission of Backend Case Study Round @BYNRY

I'd be glad to create a response combining the strengths of Response A and Response B, addressing their shortcomings, and incorporating insights from the ratings to provide an exceptional Django application for the gas utility company:

Project Structure (Bonus Points):



Explanation:

- Separation of Concerns: The project is organized into well-defined apps (accounts, core, customer_support). This promotes modularity, maintainability, and easier collaboration.
- Scalability: Each app has its own models, views, forms, and tests, allowing for independent development and testing.
- **Flexibility:** New features can be added by creating new apps without affecting existing functionality.

Core App (core):

- Models:
 - Customer: Stores customer information (name, address, account number, etc.).

- ServiceRequest: Captures request details (type, description, attachments, status, timestamps, foreign key to Customer).
- Views:
 - Customer-facing views for creating, viewing, and tracking service requests.
 - Support representative views for managing requests (updating status, assigning technicians, etc.).

Authentication (accounts):

- Models:
 - User: Extends Django's User model with additional customer-specific fields (if required).
- Forms:
 - Custom user registration and login forms (optional, if using Django's built-in authentication is insufficient).
- Views:
 - Authentication views (login, logout, password reset) for customer self-service (optional).

Customer Support App (customer_support):

- Models:
 - Technician: Stores technician information (name, qualifications, availability, etc.) (optional, if needed for technician assignment).
- Forms:
 - Forms for support representatives to update service request details, assign technicians, and communicate with customers.
- Views:
 - Views for support representatives to manage service requests in bulk, view detailed customer information, and generate reports.

Additional Considerations:

- Security:
 - Implement industry-standard security measures (e.g., secure password hashing, user permissions, input validation).
 - Regularly update Django and third-party libraries to address vulnerabilities.
- Scalability:
 - Consider using a database backend (e.g., PostgreSQL) that excels at handling high data volumes.
 - Explore caching mechanisms (e.g., Redis) for improved performance.
- Usability:
 - Design a user-friendly interface for both customers and support representatives.
 - Provide clear instructions and status updates throughout the request lifecycle.
 - Consider accessibility features for users with disabilities.

Notifications:

• Implement email or SMS notifications to keep customers informed about request status changes.

Reporting:

reate reports for management to analyze service requent provement, and track customer satisfaction.	est trends, identify areas for