Designing a scalable and reliable Django-based system for an online marketplace requires careful planning and implementation across various aspects of the system. The following is a bird's eye view of what would be the best way on how to hand out these key points:

1. Database design and optimization strategies.
    - Schema Design: The schema should be normalized in order to minimize redundancy. Properly applying concepts such as 'ForeignKey' and 'ManyToManyField' will help achieve this. Use 'UUIDField' for primary keys to avoid sequential integer IDs that can expose data structure.
    - Partitioning: Use table partitioning for very large tables to improve query performance.
    - Indexes: Make indexes on fields that are frequently employed in queries such as foreign keys and those fields that were used in a search.
    - Query Optimization: Use Django ORM efficiently, and optimize queries to avoid especially the n+1 problem.
    - Connection Pooling: To manage and reuse DB connections.
2. Caching Mechanisms.
    - Memory-based caching: This method allows for quick retrieval and lower database load by caching data in the memory of a separate caching server.
    - Template Caching: Cache rendered templates to reduce the overhead of rendering dynamic content.
    - Page Caching: Cache entire pages or views that do not change frequently.
    - Fragment Caching: Cache only parts of a page or template that are expensive to render.
3. Security Measures.
    - Encryption: Transmission of client-server data can be made safe from attackers by providing HTTPS encryption. Sensitive information should be stored securely with Django's built-in password hashing mechanisms.
    - Database Security: Use database-level encryption for sensitive field.
    - CSRF Protection: Enable CSRF protection in Django to prevent Cross-Site Request Forgery attacks.
    - XSS Protection: Use Django's built-in template system which automatically escapes output to prevent Cross-Site Scripting attacks.
4. Scalability and Performance Optimization.
    - Vertical Scaling: By increasing the capacity of a single server by adding more power.
    - Horizontal Scaling: By adding more servers to your application to distribute the load.
    - Asynchronous Processing: Use background tasks for long-running processes.
    - Database Sharding: Store a large database across multiple machines.
5. Integration with Third-Party Services
    - Payment Gateways: Stripe or Paypal to handle transactions, and QuickBooks Accounting for accounting purposes.

    - Shipping Providers: Use the API of FedEx, Transportify, Lalamove, etc. to provide real-time shipping rates and track shipments.
    - Label Generation: Generate shipping labels and handle package tracking.
6. Handling User Sessions and State Management.
    - Session Storage: Use Django’s session framework to manage user sessions.
    - Session Security: Ensure sessions are secure by using secure cookies and setting appropriate session expiration times.
    - User Preferences: Maintain the user’s specified details and configurations within the storage and confirm they are retrieved and changed quickly enough.
    - Shopping Cart: Use a combination of session storage and database persistence to manage shopping carts and user selections.