1. **User Creation (POST /api/users)**
   * ADMIN can create any role
   * MANAGER can only create STAFF
   * Creator information is recorded
2. **Role Change (POST /api/users/{userId}/role)**
   * ADMIN only
   * Prevents self-role change
3. **Get All Users (GET /api/users)**
   * All roles can view all users
4. **Get Specific User (GET /api/users/{userId})**
   * Users can view themselves regardless of role
   * Otherwise, must have permission to manage the target user's role
5. **Update User (PUT /api/users/{userId})**
   * Users can update their own info (except role)
   * Role changes are blocked (must use role endpoint)
   * Must have permission to manage the target user
6. **Delete User (DELETE /api/users/{userId})**
   * No self-deletion
   * ADMIN can delete anyone except themselves
   * MANAGER can only delete STAFF
   * STAFF cannot delete anyone (handled by @PreAuthorize)
7. **Get Users by Role (GET /api/users/role/{role})**
   * All roles can query any role

Additional protections:

* All endpoints require authentication
* Business logic checks at both controller and service levels
* Role hierarchy enforced (ADMIN > MANAGER > STAFF)
* Principle of least privilege followed
* Self-modification protections in place

Note: This implementation assumes that:

1. The UserService implements all the necessary business logic checks
2. The UserRole enum has a canManage() method that implements the role hierarchy logic
3. The User model has proper fields for tracking creator and timestamps