**Anforderungen aus deiner Beschreibung**

1. **ASP.NET Core MVC mit Razor Pages** → klassische Webarchitektur, aber auch für mobile nutzbar.
2. **SQL-Datenbank** → Benutzer, Rollen und Rechte müssen hier gepflegt werden.
3. **4 Rollen**:
   * appadmin → Superuser, verwaltet Reinigungsfirmen
   * cleaningadmin → Verwaltung innerhalb der Firma
   * cleaningstaff → Mitarbeiter der Reinigungsfirma
   * buildingowner → Kunden der Firma
4. **Login/Register mit E-Mail und Passwort** → zentrale Authentifizierung.
5. **Routing nach Rolle** → jeder Benutzer soll nach Login sein eigenes Dashboard sehen.

**Möglichkeiten in ASP.NET Core**

1. **ASP.NET Core Identity**
   * Standardlösung von Microsoft für Authentifizierung und Autorisierung.
   * Integriert User-Management, Rollen, Claims, Passwort-Hashing, 2FA etc.
   * Lässt sich direkt mit SQL (z. B. via Entity Framework Core) verbinden.
   * Vorteil: Sehr sicher, flexibel, direkt in Razor Tag Helpers nutzbar.
2. **JWT (JSON Web Tokens)**
   * Vor allem sinnvoll für **API-first-Architekturen** (z. B. wenn die Mobile App eine echte API anspricht).
   * Die mobile App und der Browser würden dann beide über einen Token mit dem Backend reden.
   * Vorteil: Trennung von API und Frontend, auch für Microservices oder externe Integrationen gut.
   * Nachteil: etwas komplexer, eher nötig wenn mehrere Clients oder externe Systeme angebunden sind.
3. **Externe Provider (OAuth2/OpenID)**
   * Z. B. Login mit Google, Microsoft, Azure AD etc.
   * Vorteil: Benutzer müssen kein eigenes Passwort merken.
   * Nachteil: Weniger Kontrolle, und du hast ja ein rollenbasiertes Modell, das stark an die Firmenstruktur gebunden ist → hier weniger passend.

**Empfehlung für dein Szenario**

Da es sich um **eine ASP.NET Core MVC Anwendung mit SQL-Datenbank** handelt, die **Rollenbasiert** arbeitet, ist die **beste Lösung ASP.NET Core Identity mit Rollenverwaltung**.

* **Login/Register** → mit Identity realisieren.
* **Rollen** → über RoleManager und UserManager pflegen (appadmin, cleaningadmin, cleaningstaff, buildingowner).
* **Autorisierung** → via [Authorize(Roles="appadmin")] oder Policies steuern.
* **Routing/Dashboards** → nach dem Login einen Redirect abhängig von der Rolle einbauen.
* **Datenbank** → Identity erzeugt automatisch Tabellen für User, Rollen, UserRoles etc., kann aber um eigene Entitäten (z. B. Kunden, Mitarbeiter) erweitert werden.
* **Optional JWT**: Falls du die Mobile App später als **separaten Client mit API** betreiben willst, kannst du **ASP.NET Identity + JWT** kombinieren. Das ist sauber erweiterbar.

**Konkreter Ablauf**

1. Registrierung/Login → über Identity (E-Mail/Passwort).
2. Rolle wird beim User in der DB gespeichert.
3. Nach Login prüfst du die Rolle → Weiterleitung auf das jeweilige Dashboard.
   * appadmin → Admin-Dashboard
   * cleaningadmin → Firmenverwaltung
   * cleaningstaff → Mitarbeiteransicht
   * buildingowner → Kundenportal
4. Autorisierung der Controller und Razor Pages mit [Authorize(Roles="...")].

**Kurz gesagt**:  
**ASP.NET Core Identity mit Rollenverwaltung ist die beste Lösung.**  
Wenn du die App später als reine API mit Mobile-Client nutzen willst, **Identity + JWT** kombinieren.