6. Communication & Notifications & Integrations

What we should cover:

- **Email templates** Already handled by CommunicationService. Supports dynamic templates in multiple languages. We'll continue building on this.
- **Push notifications** We'll extend CommunicationService to support push, so apps can notify users instantly about new messages, alerts, or features.
- **Lulu (chat)** Use CommunicationService to deliver automated system or service messages directly to Lulu.
- **SMS messages** For important or time-sensitive info (e.g. system outages or password resets), we'll use providers like **Spryng** or **Twilio**, depending on region.
- **Client updates** Use email and SMS to inform users about new features, changes, or outages. All of this goes through CommunicationService, so we don't spread logic across multiple systems.
- **Website integration (Duda)** Use Duda's API to automatically post updates (like news or features) from our system to the club websites.
- **External integrations**
  - **Meshlink (and other IoT)** and federations (e.g. **KNLTB**, **KNHB**) require sending and receiving data.
  - These will be connected using the **Hexagon architecture** so integrations don't leak into core domain logic. External APIs are wrapped in adapters that can evolve without touching the core.
- **Other tools like Mailchimp or mailboxes** can be added later either through CommunicationService or as standalone adapters.

Why high availability matters:

If CommunicationService goes down, we lose the ability to reach users—no emails, no push, no SMS, no system alerts. That's a single point of failure that would:

- Delay urgent alerts (e.g. downtime or security notices)
- Break automated flows in other services
- Prevent users from receiving login or verification emails
- Cause trust issues with clubs or end-users

What we can do about it:

To make sure CommunicationService is reliable and highly available:

- **Run multiple instances**, ideally in a load-balanced setup (e.g. Azure App Service with autoscaling or Kubernetes with HPA).
- **Use retries and fallback queues** in services that depend on it—if the service is briefly unavailable, messages can still be retried later.
- **Deploy across multiple regions** to tolerate regional Azure outages.
- **Consider multi-cloud failover** for extreme cases (e.g. mirror a minimal version on another cloud provider for redundancy).
- **Use Azure Service Bus** or other message brokers to decouple requests and avoid hard dependencies on real-time availability.