Task 1

Scalability:

1. One issue that we found is that logo uploads are processed and stored on the API server's local disk, then served from that same instance. This couples CPU- intensive image processing and storage to each web node, making horizontal scaling fragile (files won´t exist on other nodes) and turning uploads into throughput bottleneck. This forces us to use a centralized system.

Fault-tolerance:

1. All backend instances rely on a single shared MongoDB instance, making it a single point of failure. If MongoDB becomes unavailable, the backend cannot read/write persistent data, causing requests to fail and potentially crashing the backend if errors/timeouts aren't handled safely.

2. The backend also depends on a shared Redis instance. If Redis becomes unavailable and the backend assumes Redis is always reachable (e.g., for sessions/cache/queues), requests that use Redis will fail and the backend will crash during runtime or startup depending on how the Redis connection is handled.

3. Also, if MongoDB is unavailable during startup, the instance may crash. If the backend later crashes for another reason and then cannot restart because MongoDB is still unavailable, this could escalate the issue.

Short Description

Architecture: The system uses a MERN-style setup: a React + Material UI frontend (with React Router for deep linking) makes HTTP requests via Axios. Traffic is routed through a Cloudflare Zero Trust Tunnel to a Node.js/Express backend API. The backend persists core data in MongoDB, uses Redis as a fast shared cache (for rate limiting), and sends transactional emails through Mailtrap.