***1. Time Math***

* I will use UTC internally to standardize time zone while saving in database.
* When a user logs in from a specific region like **Cranbrook**, I prefer to convert times to their **local time zone** on the frontend or in API responses.
* To do this, there are libraries in JavaScript like date-fn-tz (<https://github.com/marnusw/date-fns-tz>)
* For scheduled events, I prefer to store **both UTC format and the intended time zone**, so we can support accurate scheduling across all BC regions.

***2. Logging***

* As part of a development team maintaining both internal and public-facing systems, I’ve worked extensively on implementing logging for automated process.
* Logging is essential for system health monitoring, debugging, and audit trail maintenance.
* The key objective is to balance completeness and security — capturing the right level of detail without exposing sensitive data and handling internal vs. public logs differently.

What I log:

* Error messages with stack traces
* Process start/end times and durations
* Key application events (e.g., user logins, API failures)
* IDs or references to affected entities (e.g., user IDs, post IDs)
* Warnings and alerts for unusual behavior

What I exclude:

* Personally identifiable information (PII) like names, addresses, or health records
* Sensitive credentials such as passwords or tokens
* Internal-only debug messages in production environments

Internal vs. Public Systems:

* Internal systems may include more detailed logs, including stack traces and system metrics, but must still avoid PII.
* Public-facing logs must be highly sanitized, respect user privacy, and be designed for security and audit compliance..
* Use logging levels (info, warn, error, debug) appropriately and centralize logs using tools like CloudWatch.

**3. *Technology Selection and Recommendation***

To recommend a framework, I will use a structured evaluation process:

* **Requirements Gathering:** I will with stakeholders/managers to understand key goals — including security, performance, accessibility (WCAG), integration with existing databases, and support for legislative workflows.
* **Criteria Development:** Create a framework comparison rubric focusing on:
  + Long-term support and community adoption
  + Licensing suitability for public sector use
  + Learning curve for internal staff
  + Support for RESTful APIs and authentication standards
  + Accessibility tools and documentation
* **Framework Shortlisting:** Compare frameworks such as:
  + Frontend: Next.js/React, Angular and Vue
  + Backend: Express (Node.js)
* **POC Implementation:** I will build small prototypes to assess frameworks, developer experience, testing capabilities, and integration ease.
* **Stakeholder Review:** Collaborate with managers, IT, accessibility coordinators, and business analysts to finalize a choice that aligns with policy, performance, and usability.

***4. Technical Debt***

* Technical debt means the future cost incurred when choosing a quick or suboptimal solution over a better, but more time-consuming one.

*Examples:*

* Using inline styles or hardcoded strings instead of centralized theming.
* Skipping tests to meet delivery deadlines, which later led to regressions.
* During a recent delivery, we had to implement a homepage grid where the column labels were dynamically tied to form field labels. These labels could be customized through the admin panel, requiring synchronization between form labels and homepage view.
* Due to version constraints and tight timelines, we had to prioritize what could be delivered immediately while maintaining the flexibility to complete the feature fully in a minor release.
* We delivered the homepage grid with static labels initially pulled from the database. Simultaneously, we completed the backend work for form label management. To manage the incomplete dynamic admin connection, we created a backlog ticket and delivered the label update feature in the following minor version. This allowed the labels to be updated via the admin panel and reflected in both the form and homepage views.
* This approach enabled us to meet the deadline for the main release while transparently managing the technical debt. The full feature was completed within two days post-release without regression. It also improved user flexibility for label customization and reduced the need for future code changes for label updates.

**Mitigation Strategy for Technical Debt**

1. Delivered a partial solution (static labels from the database) in the current version to meet the release deadline.
2. Created a clear backlog ticket to track the remaining work (admin panel integration for dynamic label updates).
3. Delivered the remaining functionality (admin-managed label updates) in a minor version update shortly after the main release.
4. Ensured that the partial implementation did not cause issues or require rework, keeping the architecture ready for final integration.