**1. Архітектура системи**

* **Frontend**: Angular для побудови клієнтської частини з інтерактивними інтерфейсами, що підтримують перевірку вхідних даних, виведення стану виконання задач, та взаємодію з API.
* **Backend**: ASP.NET Core Web API для обробки запитів шифрування/дешифрування, виконання задач, управління базою даних та реалізації балансування навантаження.
* **База даних**: SQLite для зберігання даних користувачів, історії задач та стану виконання поточних задач.
* **Load Balancer**: Використання Nginx або іншого балансувальника навантаження для розподілу запитів між декількома бекенд-серверами.

**2. Основні етапи роботи**

**A. Реалізація трудомістких операцій (шифрування/дешифрування)**

* Створити сервіси для шифрування/дешифрування даних. Наприклад, можна використовувати симетричне шифрування (AES) для швидкого і безпечного шифрування даних.
* Додати перевірку вхідних даних (на фронтенді) для обмеження розміру вхідних даних та кількості задач, щоб уникнути перевантаження сервера.

**B. Моніторинг прогресу виконання задачі**

* Додати **SignalR** або WebSocket для реального часу інформування клієнта про статус виконання задач (відсотки виконання, етапи процесу).
* В API додати механізм для періодичного оновлення стану задач у базі даних, наприклад, за допомогою збереження поточного стану або відсотку виконання.

**C. Обмеження трудомісткості**

* Додати обмеження на максимальну кількість невідомих або тривалість виконання задачі на сервері (наприклад, за допомогою CancellationToken або таймерів).
* У разі перевищення лімітів, система повинна повернути помилку або повідомити про відмову у виконанні запиту.

**D. Управління задачами та історія**

* Створити таблицю в базі даних для зберігання історії задач користувачів, включаючи їх статус (виконується, завершена, помилка).
* API для перегляду історії виконання задач, скасування поточних задач та запуску нових.
* Встановити обмеження на максимальну кількість активних задач для кожного користувача.

**3. Авторизація клієнта**

* Реалізувати аутентифікацію та авторизацію за допомогою **IdentityServer4** або вбудованої системи аутентифікації ASP.NET.
* Додати захищений протокол **HTTPS** для передачі конфіденційних даних.
* Реалізувати токенову аутентифікацію на базі JWT (JSON Web Token) для безпечної взаємодії між клієнтом та сервером.

**4. Балансування навантаження**

* Налаштувати **Nginx** або інший балансувальник для розподілу навантаження між двома або більше серверами застосунків (application servers).
* Реалізувати підтримку горизонтального масштабування, щоб нові сервери могли автоматично обробляти запити при збільшенні навантаження.

**5. Тестування та моніторинг**

* Перевірити, як веб-застосунок обробляє багатопоточні та трудомісткі запити.
* Використовувати **Prometheus** або **Grafana** для моніторингу стану серверів та продуктивності шифрувальних операцій.
* Стрес-тестування для оцінки продуктивності при високому навантаженні.

**6. Додаткові покращення**

* Додати **кешування** результатів для зменшення навантаження на сервери у разі повторних запитів.
* Використовувати **RabbitMQ** або іншу чергу повідомлень для обробки задач у фоновому режимі, що дозволить краще контролювати чергу задач і забезпечувати стабільну роботу при великій кількості запитів.

Цей план допоможе створити веб-застосунок, що забезпечуватиме балансування навантаження, керування трудомісткими задачами та надання функцій шифрування і дешифрування даних.