Вот план действий и схема взаимодействия между тремя уровнями системы.

1. Архитектура системы

Система состоит из трех веб-сервисов, которые взаимодействуют через WebSocket, Kafka и JSON API:

- **Прикладной уровень (Frontend + WebSocket-сервер)** – пользовательский интерфейс для обмена сообщениями.
- **Транспортный уровень (Kafka + Golang)** – отвечает за разбиение сообщений на сегменты и сборку.
- **Канальный уровень (Кодирование в [7,4]-код, эмуляция ошибок)** – отвечает за защиту данных и обработку ошибок.

2. Взаимодействие уровней

1. **Отправка сообщения**
   - Пользователь вводит сообщение или загружает файл (Frontend → WebSocket).
   - Сообщение передается на **транспортный уровень**, где разбивается на сегменты.
   - Каждый сегмент передается на **канальный уровень**, где кодируется, искажается с вероятностью **P=10%**, а затем передается обратно.
   - Сегменты передаются обратно на **транспортный уровень**, собираются в единое сообщение раз в **N=1 сек** и отправляются на **прикладной уровень**.
   - Если некоторые сегменты утеряны (**R=2%**), сообщение передается с признаком ошибки.
2. **Получение сообщения**
   - Транспортный уровень собирает принятые сегменты и передает их **прикладному уровню**.
   - Если сегмент поврежден, прикладной уровень помечает сообщение как ошибочное.

3. Детализированный план разработки

I. Прикладной уровень (Frontend + WebSocket-сервер)

Технологии: React + Redux Toolkit + MUI + Axios, WebSocket-сервер на Node.js

Шаги:

1. Разработка чата с вводом текста и загрузкой файлов.
2. Открытие WebSocket-соединения при входе, передача имени пользователя.
3. Отправка сообщений методом `Send` в JSON-формате:

```
{
  "sender": "user1",
  "timestamp": "2025-02-21T12:00:00Z",
  "payload": "Hello, world!"
}
```

4. Получение сообщений методом `Receive` (с признаком ошибки, если есть):

```
{
  "sender": "user2",
  "timestamp": "2025-02-21T12:00:05Z",
  "error": false,
  "payload": "Hello!"
}
```

5. Отображение сообщений, обработка ошибок.

II. Транспортный уровень (Kafka + Golang)

Технологии: Kafka, Docker, Golang

Шаги:

1. Принятие JSON-сообщений от прикладного уровня.
2. Разбиение на сегменты по **140 байт**. Каждый сегмент содержит:

```
{
  "message_id": "UUID",
  "total_segments": 5,
  "segment_index": 2,
  "payload": "часть сообщения"
}
```

3. Отправка сегментов в Kafka-топик → передача на канальный уровень.
4. Получение обработанных сегментов обратно, сборка полного сообщения раз в **1 сек**.
5. Передача сообщения на прикладной уровень (если потеряны сегменты, передаем ошибку).

III. Канальный уровень (Эмуляция помех, [7,4]-код)

Технологии: Python/Golang

Шаги:

1. Получение сегмента, кодирование в **[7,4]-код** (вход 4 бита → выход 7 бит).
2. Внесение ошибки в случайный бит **с вероятностью 10%**.
3. Потеря пакета **с вероятностью 2%**.
4. Декодирование:
   - Если ошибка исправлена → передача на транспортный уровень.
   - Если не исправлена → потеря сегмента.

4. Итоговая схема взаимодействия

```
[Frontend (React)]  ↔  [WebSocket-сервер]
       ⬇                          ⬆
[Транспортный уровень (Kafka, Golang)]
       ⬇                          ⬆
[Канальный уровень (Кодирование, Ошибки)]
```