

2024 年度ウェブプログラミング実習 総合実習レポート

(リアルタイム議事録)

班番号	B	研究室 名	宮下研
組	3	番号	45
氏名	松島陽也		
提出日時	1/22		
共同実習者名	菅生遥叶 吉川栄春 春日裕次		

1. 作成概要

本システム「リアルタイム議事録」はユーザ同士のテキストチャットから、チャット内容に関する議事録を生成するシステムである。本システムの特徴は、議事録をチャット画面から生成できる利便性と、議事録生成における抽出項目をユーザの任意で設定できる自由度の高さにある。本システムを開発するにあたり、利用者のターゲットはリモートによる会議を行う人に絞り、利用場面は、テキストチャットを使用した会議を想定した。

システムの基本的な動作について、図 1 より、まずユーザはログインまたは新規登録のどちらかを選択し、ログイン画面(図 2)あるいは新規登録画面(図 3)に遷移する。ログイン画面では、ユーザ名とパスワードを入力し、chat.sqlite に保管されているユーザ名とパスワードが一致すれば、ログイン完了となる。新規登録画面では、ユーザ名とパスワードを新しく設定する。設定が完了すると、ユーザはログイン完了となり、入力したアカウント情報は新たに chat.sqlite に保存され、次回ログイン時に参照される。



図 1 :index.html

図2:login_form.php

図3:register_form.php

図4:top_page.php

ログインまたは新規登録ののち、図 4 のトップページへと遷移する。図 4 より、ユーザは「ルームに参加」または「ルームを作成」を選択後、それぞれルーム参加ページ(図 5)、ルーム作成ページ(図 6)に遷移する。

図5:room_join_form.php

図6:room_create_form.php

図 5 のルーム参加画面では、ルーム名とあいことばを入力する。ルーム名とあいことばが chat.sqlite のものと一致すれば、現在開かれている、あるいは過去に開かれたルームへの参加が可能となる。図 6 のルーム作成画面では、ルーム名とあいことばを新たに設定し、ルームを作成する。この際、入力されたルーム名とあいことばは新たに chat.sqlite に書き込まれ、ルーム参加の際に参照される。



図 7: chat.html

図7はルームページである。チャット画面、抽出項目選択画面、議事録生成画面で構成されており、ここで他のユーザとチャットから議事録生成まで行える。

チャット画面はユーザ同士の会話が非同期通信で表示され、テキストでのチャット入力ができる。抽出項目画面では ChatGPT-4 にチャット内容から要約してほしい項目をユーザが複数入力する。要約画面では ChatGPT-4 が要約項目に沿った内容を要約する。要約画面には要約削除と会話削除のボタンが存在する。要約と会話内容はデータベースに書き込まれる。会話内容と要約は既存のルームに参加した際に、データベースから読み込まれ、表示される。要約削除と会話削除では、データベースから会話履歴、要約が削除される。

図2はシステム全体の画面遷移図である。

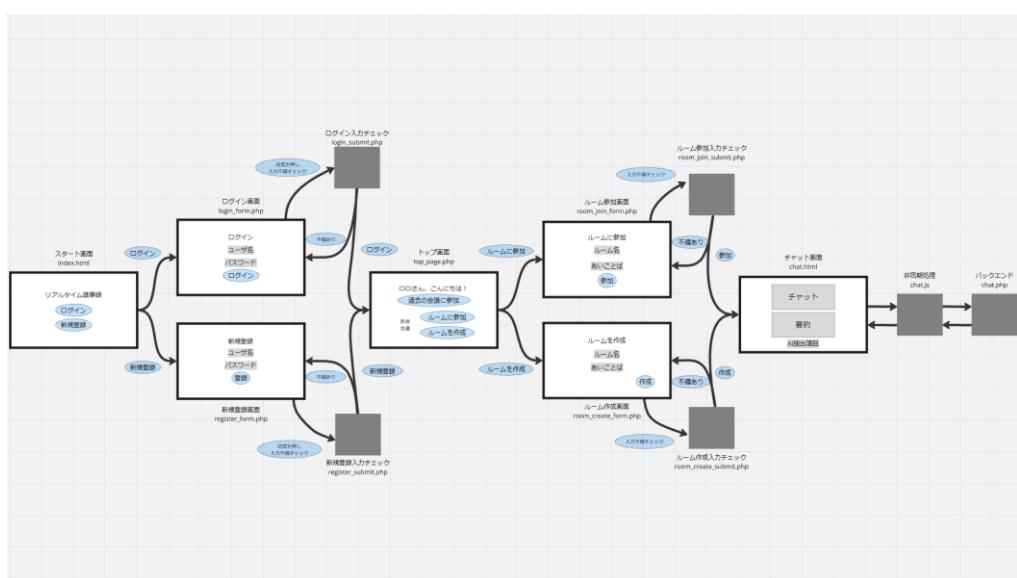


図8:システム全体の画面遷移図

図9は chat.php における ER 図を示している。

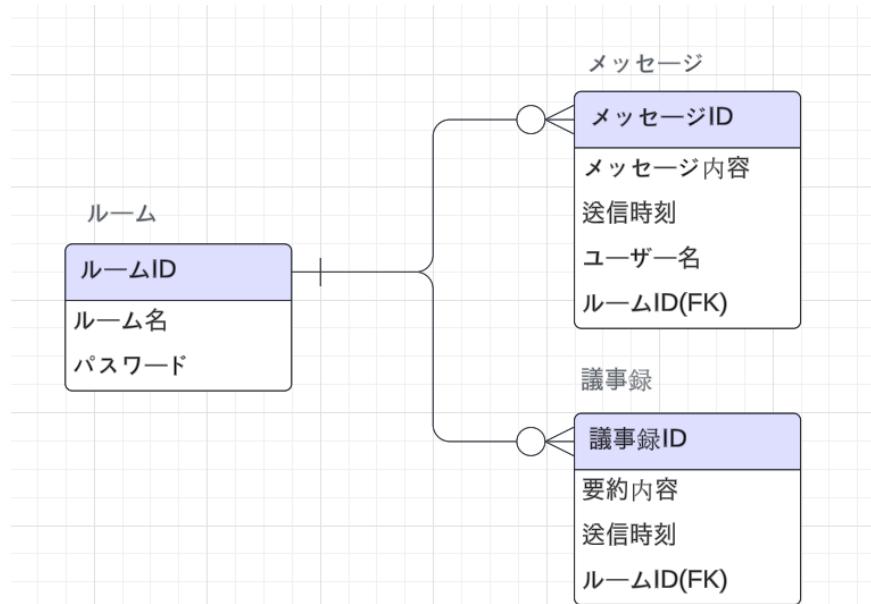


図9:chat.php における ER 図

図10～14は chat.sqlite の各テーブルのスキームを表している。

1. user

カラム名	データ型	制約
id	INTEGER	PRIMARY KEY AUTOINCREMENT
username	TEXT	
password	TEXT	

図10:user テーブルのスキーム

2. sqlite_sequence

カラム名	データ型	制約
name	TEXT	
seq	INTEGER	

図11:sqlite_sequence テーブルのスキーム

3. room

カラム名	データ型	制約
id	INTEGER	PRIMARY KEY AUTOINCREMENT
roomname	TEXT	NOT NULL
password	TEXT	NOT NULL

図12:room テーブルのスキーム

4. messages

カラム名	データ型	制約
id	INTEGER	PRIMARY KEY AUTOINCREMENT
room_id	INTEGER	FOREIGN KEY REFERENCES room(id)
username	TEXT	NOT NULL
message_text	TEXT	NOT NULL
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

図13:messages テーブルのスキーム

5. summaries

カラム名	データ型	制約
id	INTEGER	PRIMARY KEY AUTOINCREMENT
room_id	INTEGER	UNIQUE, FOREIGN KEY REFERENCES room(id)
summary_text	TEXT	NOT NULL
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

図14:summaries テーブルのスキーム

2. 担当箇所

表 1: 担当箇所の表

項目	担当者	備考
サーバ、利用者管理	春日、菅生	PHP のプログラミング、ログイン処理
ユーザインタフェース	菅生、吉川、松島	CSS 作成、議事録抽出項目入力欄、チャット画面作成
通信機能	松島	PHP・JS のプログラミング
議事録生成機能	吉川	GPT リクエストデータ作成
プレゼンテーション	松島、吉川、春日	スライド作成・発表
画面遷移図、ER 図作成	松島、菅生、春日	画面遷移図作成、プログラム作成時の設計図、ER 図
データベース	春日	SQL の設計、更新、ルームデータの読み込み、保存

自分は表1にある通り、PHP・JS のプログラミングとプレゼンテーションスライドの作成・発表、画面遷移図の作成を行った。編集したプログラムを編集箇所が多い順に並べると、Chat.js, Chat.php, Chat.html となる。

本システムにおいて、私が担当したのは非同期処理の実装に関する全ての部分である。これらのプログラムにより、非同期処理システムを実装し、ポーリング方式で通信を行うようにした。以下に、実装の詳細を説明する。

2.1. 非同期処理の概要

非同期処理とは、ある処理の完了を待たずに次の処理を実行できる仕組みである。本システムでは、ユーザインタフェースの応答性向上と長時間かかる処理のバックグラウンド実行、リアルタイムなデータ更新の実現を目的として非同期処理を導入した。

2.2. 実装方法

非同期処理の実装には、JavaScript の Async/Await 構文を使用した。これにより、コードの可読性を保つつつ、複雑な非同期処理を実現した。以下に、システムの中核となる非同期関数の例を示す。

```
async function Update() {
  try {
    const response = await fetch('chat.php', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ action: 'get_update' })
    });
    const data = await response.json();
    if (data.success) {
      messagesDiv.innerHTML = '';
      data.messages.forEach(displayMessage);
    } else {
      console.error(data.error || '更新に失敗しました。');
    }
  } catch (error) {
    console.error('エラー:', error);
  }
}
```

この関数は、100ミリ秒ごとに実行され、サーバから最新のメッセージを取得し、表示を更新する。

また、ポーリング方式による通信を実現するため、`setInterval`関数を使用して定期的に`Update`関数を呼び出している：

```
setInterval (Update, 100);
```

この実装により、クライアントは 100 ミリ秒ごとにサーバに新しいメッセージがないか問い合わせる。

2.3. エラーハンドリング

非同期処理におけるエラーハンドリングは、try-catch 文を使用して実装した。各非同期関数内でエラーが発生した場合、適切にキャッチし、コンソールにエラーを出力するとともに、必要に応じてユーザにアラートを表示する。

2.4.パフォーマンス最適化

非同期処理の導入により、システム全体のパフォーマンスが向上した。具体的には、メッセージ送信や要約生成などの処理をバックグラウンドで実行可能にし、ユーザインターフェースのブロッキングを防止するとともに、リアルタイムなデータ更新による即時性を向上させた。

特に、`updateSummary`関数では、長時間かかる可能性のある要約生成処理を非同期で行うことで、ユーザエクスペリエンスの低下を防いでいる。

2.5.セキュリティ対策

非同期通信におけるセキュリティ対策として、以下の措置を講じた。クロスサイトスクリプティング対策として、ユーザ入力のエスケープ処理を行い、CSRF 対策として、セッション管理を実装した。

2.6.苦労した点

非同期処理の実装において、いくつかの困難に直面した。まず、非同期通信特有のエラーパターンへの対応に苦心した。ネットワークエラーやタイムアウトなどの問題に適切に対処するため、エラーハンドリング機構の設計と実装に多くの時間を費やした。try-catch 文を効果的に配置し、ユーザに分かりやすくエラーメッセージを表示する仕組みを構築するのは容易ではなかった。

次に、ポーリング方式による頻繁な通信がサーバに与える負荷を最小限に抑えつつ、リアルタイム性を確保するバランスを取るのに苦労した。適切なポーリング間隔の設定や、必要最小限のデータのみを転送する最適化を行うことで、この課題に取り組んだ。

非同期処理のデバッグは同期処理と比べて複雑で、エラーの原因特定に時間を要した。特に、タイミング依存のバグの再現と修正には多大な労力を要した。これらの問題に対処するため、綿密なテストケースの設計と実行を行った。

最後に、非同期処理によるデータ更新をユーザにどのように提示するかという点で苦心した。新しいメッセージの到着をスムーズに表示しつつ、ユーザの現在の操作を妨げないインターフェースの設計に腐心した。ユーザエクスペリエンスを損なわずにリアルタイムな情報更新を実現するため、様々なアプローチを試行錯誤した。

2.7.まとめ

以上が、本システムにおける非同期処理の実装の詳細である。この実装により、システムの応答性と効率性が大幅に向上し、ユーザエクスペリエンスの改善に貢献した。今後の課題としては、WebSocket の導入によるリアルタイム性のさらなる向上が考えられる。

3.参考文献

1) * API の使用及びコード生成に使用

OpenAI. "ChatGPT".GPT-4. (<https://chatgpt.com/> ,2024 年 12 月参照)

2) * 画面遷移図作成に使用

miro.(<https://miro.com/ja/> , 2025 年 1 月参照)

3) * ER 図作成に使用

lucid.(<https://lucid.co/ja> , 2025 年 1 月参照)