データベース及び演習 （2023年7月31日）

水谷祐生[1,a)]

概要：本稿は，情報処理学会論文誌ジャーナルに投稿する原稿を執筆する際，および論文採択後に最終原稿を準備する際の注意点等をまとめたものである．大きく分けると，論文投稿の流れと，LaTeX と専用のスタイルファイルを用いた場合の論文フォーマットに関する指針，および論文の内容に関してするべきこと，するべきでないことをまとめたべからずチェックリストからなる．本稿自体も LaTeX と専用のスタイルファイルを用いて執筆されているため，論文執筆の際に参考になれば幸いである．

Database and Exercises (version 2012/10/12)

Yusei Mizutani[1,a)]

1. 機能概要

このアプリケーションには

- 「作成者名」「部屋名」「部屋の説明」を入力した後、「作成」ボタンを押すことで部屋を作成する機能
- 部屋の説明欄を編集する機能
- タグを作成する機能
- 作成したタグを部屋の参加者に割り当てる機能
- 参加者を追加する機能
- 参加者を削除する機能
- 参加者カードの「編集」を押すことで「名前」または「コメント」を編集する機能
- 部屋の参加者の受付をするかしないかを決める機能

といった機能が実装されている．

2. 利用技術

2.1 Next.js

2.1.1 Next.js とは

Next.js とは React をベースに開発された、フロントエンドフレームワークである．

Next.js では画像・レンダリングを最適化する。最適化するメリットとしてページの読み込み速度が高速になることである. また、読み込み速度の高速化は SEO にも効果がある. このように Next.js を利用することで、Web ページ閲覧時のユーザー体験を向上させる、作成したページを誰かに見てもらいやすくなるといったメリットがある．

2.1.2 React とは

React とは Facebook 社が開発した Web サイト上の UI パーツを構築するための JavaScript ライブラリである. また、JSX 記法と呼ばれる構文が採用されていて、これにより一つのファイル内で HTML タグに対する動きを操作することができる. 同様の JavaScript のライブラリとして Vue.js、Angular などが挙げらるが、React は中でも世界的に圧倒的な導入率を誇ってている。

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，情報処理大学
Presently with Johoshori Uniersity
[a)] joho.taro@ipsj.or.jp

2.2 ChakraUI

Chakra UI とは、UI コンポーネントライブラリの 1 つで、自前で CSS を書かなくてもパラメータ指定でスタイルを記述でき、デザインに一貫性を持たせることができる。再利用可能なコンポーネントも多く用意されていて、アラートダイアログやドロワーメニュー、トーストなど自前で作ろうとすると大変なものも簡単に実現できる。また、レスポンシブ対応も容易に行うことができるため誰でも簡単に洗練されたデザインの UI を構築することが可能である.

2.3 TypeScript

TypeScript とはオープンソースかつ、JavaScript を拡張して開発されたプログラミング言語であり、Microsoft によって開発された.

基本的な文法は JavaScript と大した変化はないが、大きな違いとして型付けができるかどうかが挙げられる.

従来までの JavaScript には型付けがない、つまり動的型付けといって実行時にプログラム側が勝手に数値型、文字型を判定するため実行時のエラーの存在に気づかず思わぬバグに繋がることがあった. この問題を解決するため TypeScript では静的型付けといって実行前にプログラマーがあらかじめ型を設定できるようにした. また静的型付けによって、変数の説明ができるため大規模な開発においてソースコードの可読性という観点からも恩恵を受けることができる.

2.4 Go

2.4.1 Go 言語とは

Go 言語とは、シンプルかつ高速な処理が可能なプログラミング言語であり、Google 社によって開発された. コードをシンプルかつにそして高速に動作させるというコンセプトの元に開発されたため、他の言語には実装されている機能、例えば「継承」や「ジェネリクス」などが実装されていない. そのため、誰が書いても似たようなコードになり、アプリケーションの管理を簡単にすることができる.

2.4.2 Gin とは

Gin とは Go 言語の中でも人気のある Web フレームワークである.

Gin には高速なパフォーマンス、JSON のリクエストのバリデーション、ルーティングのグループ化、エラー管理、組み込みのレンダリング、といった特徴があるため、Gin なしで開発するよりもより高速、簡単に Web アプリケーションや API サーバーを開発することができる.

2.5 Docker

Docker とは、Docker 社が開発するコンテナ型のアプリケーション実行環境のことである.Docker を使うことで、1 台のサーバー上で様々なアプリケーションを手軽に仮想化・実行できるようになる.

Docker はコンテナ環境をファイルで設定することができるため、Docker ファイルを共有することで開発環境を揃えることが大変であるチーム開発時に大きな有用性がある. また、一台のサーバーに複数のアプリケーション実行環境を作成することができるため、デプロイのしやすさや、サーバーの節約にも貢献することができる.

2.5.1 API

API とは「Application Programming Interface」の略である. 言葉通りに意味を解釈すると、アプリケーションをプログラミングするためのインターフェースという意味である. 簡単に説明すると、API とはソフトウェアに API という外部とやりとりする窓口であり、外部アプリとコミュニケーションや連携ができるものである.

3. システム設計

3.1 システム概要

今回のアプリのシステム構成図は図 1 のようになっており、UI 描写専用の Next.js のサーバーと MySQL サーバーとやり取りを行う Go サーバー今回はフレームワークとして Gin を使用しているため Gin サーバーの二つが連携することによって動作するようになっている. 具体的な連携方法として、HTTP メソッドでやり取りをを行っており、今回は Next.js サーバーが GET、POST、PUT、DELETE メソッドにパラメーターやボディーに変数を乗せて Gin サーバーにリクエストを送り、バックエンド側でそれに応じたデータベースに対する操作を行い、最終的な結果を Next.js サーバー、つまりフロントエンドにレスポンスとして返している.

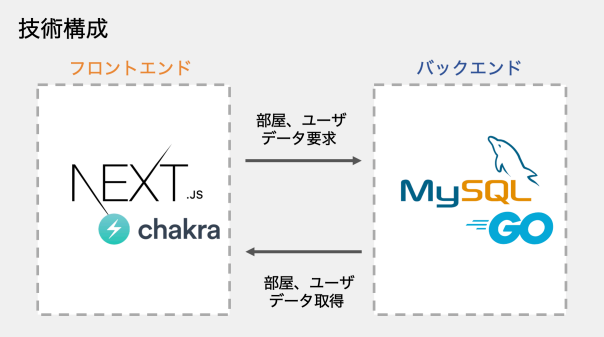

図 1 システム構成図

3.2 技術選定の意図

3.2.1 Next.js

Next.js はデフォルトでサーバーを持っているため、create react app を利用して開発した際に設定しなくてはならない Webpack や babel の記述の負担を減らすことができ

るため採用した.

3.2.2 ChakraUI

ChakraUI は CSS を記述しなくとも洗練されたデザインの UI コンポーネントを扱うことができるため、フロントエンド側でのデータ処理やクリック時の処理に時間を咲くことができる. すなわち、システム的な機能実装に集中することができるため採用した.

3.2.3 Go

他にも Python、Java などが候補に上がったが、中でも Go 言語は Gorm といった ORM マッパーや、Gin といった強力な Web フレームワークが備わっているため、API サーバーを作成するために便利なパッケージが備わっている. それゆえに、Go 言語を採用した

3.2.4 MySQL

今回のアプリの性質上、RDB を用いると最も容易にデータ管理を行うことができると考えたため採用した.

3.3 画面遷移

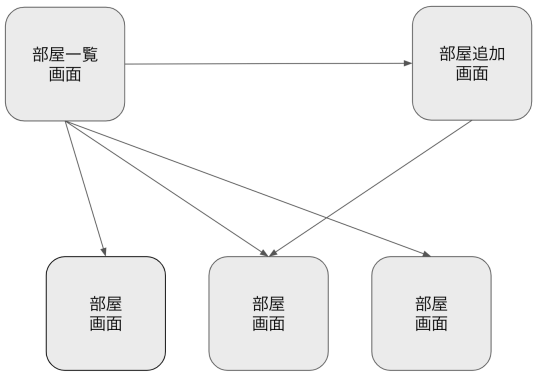

図 2 画面遷移の図

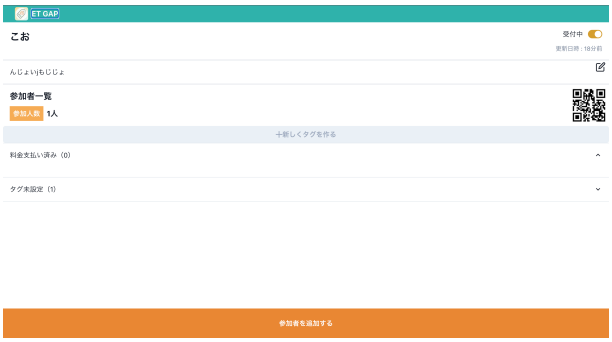

図 3 部屋画面の図

図 2 は今回のプログラムにおける画面遷移を表したものであり、以下にその詳細を記す.

3.3.1 部屋作成の動き

(1) 「部屋一覧画面」で「新しく部屋を作る」ボタンをおして「部屋追加画面」に遷移する
(2) 「部屋追加画面」で作成者名、部屋名、部屋の説明を入力して「作成」ボタンをおす
(3) ボタンを押したと同時に部屋を新規作成、部屋画面に遷移する

といった流れになっている.

3.3.2 部屋参加の動き

(1) 部屋に入っている人から「部屋画面」に表示されている QR コードまたは URL を共有してもらう
(2) 「部屋画面」の「参加者を追加する」ボタンを押すと、ドロワーが表示される
(3) ドロワーにある名前、コメントを入力してから「参加する」ボタンを押す

といった流れになっている.

3.3.3 タグ付の動き

(1) 「新しくタグを作る」を押す
(2) ドロワーが表示される
(3) ドロワーの「タグ」に作成したいタグ名をつける
(4) 図 3 のユーザーカードにある「編集」ボタンを押す
(5) 押すと「名前」、「コメント」、「タグ」の入力ができるため「タグ」の欄に作成したタグ名を入れる

といった流れになっている.

3.3.4 参加者削除の動き

(1) 図 3 のユーザーカードにある「削除」ボタンを押す
(2) 確認用のドロワーが表示される
(3) ドロワーにある「はい」ボタンをクリックする

といった流れになっている.

3.4 Docker 内の処理

今回のアプリケーションはバックエンドを全て Docker で行なっている. このセクションでは Go と MySQL コンテナについて、docker-compose.yml と Dockerfile を参照しながら説明をする

3.4.1 Go コンテナ

```
go:
  container_name: Database_And_Exercises_Go
  build:
    context: ./Docker/go
    dockerfile: Dockerfile
  stdin_open: true
  tty: true
  env_file:
    - ./Docker/go/.env
  volumes:
    - ./go:/go/src/app
  ports:
```

```
      - 8080:8080
    depends_on:
      - "mysql"
```

上は docker-compose.yml の中でも Go コンテナに関する記述を抜粋したものである. 以下、それぞれの部分に関する説明を行う.

(1) container_name でコンテナの名前を指定する. 今回は「Database_And_Exercises_Go」
(2) build の context に Dockerfile のあるパスを docker-compose.yml から見たときのパスを入力し、docekrfile に読み込む Dockerfile のファイル名を指定する
(3) stdin_open とはコンテナ内のエラーを標準出力として表示するかを boolean で指定する要素である
(4) tty とは擬似端末 (キーボードによる入力) をコンテナに結びつける設定である
(5) env_file で環境変数として読み込む env ファイルのパスを指定している
(6) volumes で「:」よりも右にあるパス以下のファイル、フォルダを「:」よりも左にあるパス以下に全てコピーする
(7) ports で Gin サーバーのポート番号を指定する. 今回は 8080 番を指定した
(8) MySQL コンテナよりも早く Go コンテナが立ち上がっていると実行時に DB に接続ができなくエラーが発生するため depends_on で MySQL コンテナが立ち上がってから Go コンテナを立ち上げるようにする

```
FROM golang:1.19.3-alpine
RUN apk add --update &&  apk add git
RUN mkdir /go/src/app
WORKDIR /go/src/app
ADD . /go/src/app
CMD ["go", "run", "./cmd/main.go"]
```

上のコードは Go コンテナに関する Dockerfile である. 以下、それぞれの部分に関する説明を行う.

(1) FROM 以下に取得したい Docker イメージを取得する. 今回は Go 言語に関するイメージを取得する
(2) RUN コマンドでビルド時に実行したいコマンドを指定する. 今回は git のインストールとディレクトリを作成した
(3) WOEKDIR でコンテナに入って操作するとき、一番最初に指定するパスを指定する
(4) ADD でコンテナに指定したフォルダやファイルをコピーする
(5) env_file で環境変数として読み込む env ファイルのパスを指定している
(6) CMD でコンテナを立ち上げたときに実行するコマンドを指定する. 今回はコンテナ起動時に Gin サーバーを起動させたいため、main.go を実行するようにしている

3.4.2 MySQL コンテナ

```
  mysql:
    container_name: Database_And_Exercises_DB
    build:
      context: ./Docker/mysql
      dockerfile: Dockerfile
    ports:
      - "3306:3306"
    volumes:
      - ./Docker/mysql/init:/docker-entrypoint-initdb.d
    environment:
      MYSQL_ROOT_PASSWORD: admin
```

(1) container_name でコンテナの名前を指定する. 今回は「Database_And_Exercises_DB」
(2) build の context に Dockerfile のあるパスを docker-compose.yml から見たときのパスを入力し、docekrfile に読み込む Dockerfile のファイル名を指定する
(3) ports で MySQL サーバーのポート番号を指定する. 今回は 3306 番を指定した
(4) volumes では MySQL に初期化用の sql ファイルのパスを指定して、それを元にコンテナ立ち上げ時、データベースとテーブルを作成するようにしている
(5) environment で MySQL コンテナ内のルートパスワードを設定している

```
FROM mysql:latest
COPY mysqld_charset.cnf /etc/mysql/conf.d/mysql_charset.c
```

上のコードは MySQL コンテナの Dockerfile である.

以下、それぞれの部分に関する説明を行う.

(1) FROM は Go コンテナ同様、インストールするイメージを指定している.
(2) COPY でインストールしたイメージにローカルのファイルをコピーしている. 今回は MySQL を日本語対応させるためのカセットをコピーしている

3.5 データベース設計

今回作成したアプリケーションでは二つのテーブルが存在する. 一つ目は、部屋情報を管理するためのテーブル、部屋の参加者情報を管理するためのテーブルこれら二つのデータを MySQL を通して行うことでシステムを動かしている.

以下に、それぞれのテーブルについて記述していく.

3.5.1 rooms テーブル

```
CREATE TABLE data_sets.rooms(
    id INT NOT NULL AUTO_INCREMENT,
    room_name VARCHAR(20),
    member_amount INT,
```

```
    summary VARCHAR(50),
    is_open boolean,
    last_update VARCHAR(25),
    tags VARCHAR(1023),
    room_maker VARCHAR(15),
    PRIMARY KEY (id)
) ENGINE = InnoDB;
```

上のテーブルは、部屋情報を管理するためのテーブルである.

部屋情報とメンバー情報を結びつけるため、今回はプライマリーキーとして部屋 ID を使用していて、Go 側で部屋にどんなメンバーが格納されているか結合することができる. また、タグの管理は、別のテーブルを用意しているわけではなく、tags にカンマ区切りの文字列として格納されていてフロントエンドでパースするようにした.

3.5.2 members テーブル

```
CREATE TABLE data_sets.members(
    id INT NOT NULL AUTO_INCREMENT,
    room_id INT NOT NULL,
    member_name VARCHAR(15),
    comment VARCHAR(30),
    tag VARCHAR(15),
    PRIMARY KEY (id),
    FOREIGN KEY room (room_id) REFERENCES rooms (id)
) ENGINE = InnoDB;
```

上の SQL 文はメンバーのデータを格納するために定義されたテーブルである.

このテーブルはフォーリンキーとして rooms テーブルの ID を使用していて、この ID を元にすることでそれぞれのメンバーがどこの部屋に所属しているか判別できるようにしている.

3.6 HTTP リクエスト

このセクションでは、このアプリで用いている HTTP メソッドのエンドポイントとバックエンドで行われる処理について説明する.

3.6.1 GET メソッド

`/api/room?roomId=`

部屋情報を取得する際に使用する. このエンドポイントは roomId を URL にパラメータとしてバックエンドに送ることで、roomId に対応した部屋情報と部屋に所属しているメンバーを全て返す処理をしている.

3.6.2 POST メソッド

`/api/room`

新しく部屋を作成するときに使用される. POST メソッドでこのエンドポイントを用いて、リクエストボディに下記の形でデータを送ることでデータベースに新しくレコードを追加し、部屋を作成する. その後、バックエンドサーバーからのレスポンスとして作成した部屋の情報を返す.

```
type Room = {
  roomId: number;
  roomName: string;
  memberAmount: number;
  summary: string;
  isOpen: boolean;
  lastUpdate: string;
  members: Array<Member>;
  roomMaker: string;
  tags: string;
};
```

`/api/room/:roomId/member`

部屋のメンバーを追加したいときに使用される POST メソッドでこのエンドポイントを用いて、リクエストボディに下記の形でデータを送ることでデータベースに新しくレコードを追加し、部屋を作成する. その後、バックエンドサーバーからのレスポンスとして作成した部屋の情報を返す. ただし、tag には空文字が入力される.

```
type Member = {
  memberId: number;
  name: string;
  comment: string;
  tag: string;
};
```

3.6.3 PUT メソッド

`/api/room/:roomId`

部屋の情報、例えばメンバー受付中や部屋の説明などを変更したときに使用される. パラメータに変更したい部屋 ID を入れた上で、リクエストボディに Room 型の値を送ることでデータベースの部屋の情報を書き換える. その後、バックエンドサーバーからのレスポンスとして変更後の部屋情報を返す.

`/api/room/member/:memberId`

メンバーの情報を変更したときに使用される. パラメータに変更したい部屋 ID とメンバー ID を入れた上で、リクエストボディに Member 型の値を送ることで、データベースの部屋の情報を書き換える. その後、バックエンドサーバーからのレスポンスとして変更後の部屋情報を返す.

3.6.4 DELETE メソッド

`/api/room/:roomId`

部屋の情報を削除したときに使用される. パラメータに変更したい部屋 ID を入れて送ることで部屋の情報をデータベースから削除する. その後、バックエンドサーバーからのレスポンスとして削除した部屋情報を返す.

`/api/room/member/:memberId`

メンバーの情報を削除したときに使用される. パラメータに変更したい部屋 ID とメンバー ID を入れて送ることで目

的のメンバーを探索し、部屋の情報をデータベースから削除する. その後、バックエンドサーバーからのレスポンスとして削除した部屋情報を返す.

3.7 ディレクトリ構成

3.7.1 フロントエンド

フロントエンドでは主に components、hooks、pages ディレクトリ以下のファイルを記述した.

分け方としては components 以下に UI 描写に関係する TSX ファイルを配置し、バックエンドに HTTP リクエストを送ったり、受け取った値をパースしたりするロジックの部分は hooks ディレクトリ以下に TS ファイルとして記述した.

また components 以下のファイルはアトミックデザインを採用しており、再利用しやすい、抽象度の高い UI コンポーネント設計を意識してコーディングを行った.

3.7.2 バックエンド

バックエンドでは、大きく Docker ファイルを管理する Docker ディレクトリ、ソースコードを管理する go ディレクトリを作成した.

また、今回のバックエンドサーバーはフロントエンドとデータベースを繋ぐ API として利用しているため、単純なシステムでよく用いられる MVC を採用して model、controller とディレクトリを分けながらコーディングを行った.

3.8 各画面設計

各画面は frontend/web ディレクトリの pages ディレクトリ以下の TSX ファイルが componets ディレクトリ以下の TSX ファイルをインポートしながらを生成されていている.

今回のアプリケーションでは誰でも簡単に操作できることを目的としていたため、ボタンの配置を大きくしたり、大切な操作はわかりやすく暖色を使ったりとわかりやすく設計を心がけていた.

3.8.1 ホーム画面

ホーム画面は index.tsx より出力されている

この画面では過去に閲覧履歴のある部屋が一覧で表示されていて、部屋のコンテンツをクリックするとその部屋の画面に遷移するようになっている. 閲覧履歴の保存方法としてはローカルストレージを利用し、部屋 ID を格納することで閲覧することができるようにしている.

3.8.2 部屋作成画面

部屋作成画面の描写には、room_building.tsx を用いている

ここでは、ユーザーからの入力として部屋の作成者名、部屋名、部屋の説明を受け付けた後、「作成」ボタンをクリックすることで入力された内容に応じた部屋が画面に遷移する.

また、入力された値は react-hook-form の useForm を用いて管理していてバリデーションの実装など開発を容易にするようにした.

3.8.3 部屋画面

部屋画面は [roomId].tsx より出力されている

この画面では部屋のメンバーの追加、削除、部屋説明の変更、タグの追加削除、メンバーの追加を受け付けるかどうかの変更を行うことができる.

メンバーの表示方法としては、バックエンドからメンバーの情報は配列として送られてくるため、それをフロントエンドで map を用いて要素を取り出すことで表示するようにしている.

4. 実装

4.1 実装環境

表 1 用いたツール名とそのバージョンの表

名前	バージョン
OS	macOS Venture バージョン 13.4
Node.js	20.1.0
Docker Desktop	4.16.2 (95914)
next	13.1.6

今回は以上の表のような環境で動作確認を行なった.

4.2 環境設定

4.2.1 Docker

(1) 公式サイトから Docker をインストールする
(2) Docker を起動させるとメニュバーに Docker のアイコンが表示される
(3) 表示されたアイコンをクリックして一番上に「Docker Desktop is running」と緑色の丸と一緒に表示されたいることを確認

といった流れになっている.

4.2.2 Node.js

(1) Node.js の公式サイトに飛ぶ
(2) 推奨版の項目から自分の使用している PC の OS にあったインストーラーをダウンロード

4.3 実行方法

今回のアプリケーションを動作させるためにはフロントエンド、バックエンドのサーバー両方を立ち上げる必要がある.

4.3.1 フロントエンド

(1) プロジェクトのルートディレクトリで「cd frontend/web」を実行する
(2) 「npm run dev」で Next.js のサーバーを立ち上げる
(3) 「http://localhost:3000/」にアクセスする

ここで、Web 画面が表示されていれば問題なくフロントエンドサーバーは立ち上がっている.

4.3.2 バックエンド

(1) プロジェクトのルートディレクトリで「cd backend」を実行する
(2)「docker compose build」で Go と MySQL 二つの Dockerfile をもとに Docker イメージを取得する
(3)「docker compose up」で MySQL コンテナと Go コンテナを立ち上げる
(4)「http://localhost:8080/」にアクセスする

ここで、「404 page not found」が表示されていれば問題なくバックエンドサーバーは立ち上がっている. また、「docker compose up」をした時、

4.4 動作検証

画面遷移の説明の際に用いた図 2 の手順を踏んで行なった. まずは一般会員に関する動作検証を行なった.

(1) ホーム画面で「新しく部屋を作る」ボタンをクリックした時部屋作成が面に遷移するか検証
(2) 必要項目を入力せずに「作成」ボタンを押した時バリデーションがされるかを検証
(3) 登必要項目を入し、「作成」ボタンを押した時入力された値を元に作成された部屋画面に遷移するか検証
(4) 部屋画面で「参加者を追加する」ボタンを押した時、モーダルが表示されるか検証
(5) 表示されたモーダルにバリデーションが機能しているかを検証
(6) 表示されたモーダルの「参加する」ボタンを押すことでメンバーが追加されいるか検証
(7) 受付中の横にあるボタンを押すことで受付か締切かを切り替えることができるかを検証
(8) 受付中の横のボタンを押すことで、「参加者を追加する」ボタンの色が切り替わるかを検証
(9)「新しくタグを作る」ボタンをおすとモーダルが表示されるか検証
(10)受付中の横にあるボタンを押すことで受付か締切かを切り替えることができるかを検証
(11)受付中の横のボタンを押すことで、「参加者を追加する」ボタンの色が切り替わるかを検証
(12)「新しくタグを作る」ボタンをおすとモーダルが表示されるか検証
(13)タグ追加のモーダルでバリデーションが機能しているか検証
(14)タグ追加のモーダルに必要項目を入力した上で「追加」ボタンを押すとタグが追加されるか検証
(15)メンバーの要素の右にある「削除」ボタンを押すとモータルが表示され、削除確認を行うとメンバーが部屋から削除されるかを検証
(16)メンバーの要素の右にある「編集」ボタンを押すとモータルが表示され、名前、コメント、タグを変更することができるか検証
(17)左上のタイトルを押すとホーム画面に遷移できるかを検証
(18)タグ追加のモーダルでバリデーションが機能しているか検証
(19)タグ追加のモーダルで変更内容を保存した時、UI に変更が反映されるか検証
(20)タグ追加のモーダルでバリデーションが機能しているか検証

5. まとめ

今回制作したアプリケーションでは二次会を簡単に開催しようといったコンセプトのもとで行なった. しかし、人数を追加するために QR コードかリンクを踏んでもらうかのどちらかでしか行うことができなかった. であるため、今後の展望としては Web NFC を用いて携帯同士を近づけることで部屋画面に遷移できるようにしたり、Web BLE で近くにいる人に一斉に部屋に参加するかどうかを聞けるような機能を実装していきたいと考えている.