2022年実務訓練T1チーム(会議室予約システム)

メンテナンス向けマニュアル

はじめに

この文書は、2022年の実務訓練学内PBLにおいて、T1チームが作成した会議室予約システムを(もし)メンテナンスすることがあったときのために各ファイルについて解説を残すためのものです。この文書が助けになれば幸いです。

開発環境

- **Flask** : ver2.0.2
- **Python** : ver3.10.2
- **SQLite** : ver3.37.2
- **Bootstrap** : ver5.0
- **JQuery** : ver3.5.1

ブランチについて

https://github.com/T1-2022/yoyaku.git GitHubには開発上で様々なブランチが作られたので、それについての説明を記します。

develop

defaultのブランチです。最終的にこのブランチに成果物が統合されていきます。

main

一番最初に作られたひな形のブランチです。

基本的にはおのおの好きにブランチを作って作業し、完成したらdevelopにマージするという方針をとっています。

作業方法

- ベースとなるブランチは「develop」

```
git checkout develop
```

- developを最新状態にする

```
git pull origin develop
```

- branchを分ける

```
git branch 作業ブランチ名
```

- 作業ブランチに移動する

```
git checkout 作業ブランチ名
```

- 作業を行う
- 変更を追加

```
git add -A
```

- 変更をコミット（issue番号つけると分かりやすい，つけなくても可）

```
git commit -m "コメント #issue番号"
```

- ローカルでマージを行う(developを取り込み整合性を合わせる)

```
git merge develop
```

- Conflictが発生したら、Conflictを解消する
- 変更をリモートに送る

```
git push origin 作業ブランチ名
```

- 作業が完了したらプルリクエストをdevelopに送る

  コメントに「close #issue番号」と記述すれば、自動的にissueが閉じられるはず

ディレクトリ構成

開発に使用したFlaskというフレームワークでは、ディレクトリの構成がある程度決まっています。以下では各ディレクトリについて説明します。

.idea

主にxmlファイルを入れるためのディレクトリです。

app

アプリケーション全体のフォルダです。

参照

デザイン案やデータベースの仕様書など参考資料を入れています。

app/migrations

データベースの更新等に用いるマイグレーションファイルを含むフォルダ。 基本的には、ここを編集する必要はなく、データベースのメンテナンス に示す方法を用いれば、データベースの更新を行うことができる。

参考：Flask-Migrate

app/models

データベースを構築するpythonファイルを入れています。

app/static

cssやjavascriptを入れるためのディレクトリです。

app/static/css

各画面でのデザインを設定するjavascriptファイルを配置するフォルダ。 本プロジェクトでは一部のデザインにbootstrapを用いている。

app/static/js

各画面での動的な動きを設定するjavascriptファイルを配置するフォルダ。 一部のファイルでJQueryを使用した記述がある。

app/static/img

画像ファイルを配置するフォルダ。

app/static/vendor

JQuery等の外部ライブラリの設定用ファイルが配置されている。

app/static/vendor/jquery

JQueryを使用するためのファイルを設置。

app/static/vendor/animsition

animsition提供の ページ遷移を設定するフォルダ。

app/static/images/icon

ファビコンを配置したフォルダ。

app/static/fonts

フォントやアイコン等のデザインファイルを配置するフォルダ。

app/static/fonts/font-awesome-4.7.0

font awesome提供のWebアイコン/。

app/static/fonts/iconic

Material Design Iconic Font提供のWebアイコン。

app/static/fonts/poppins

Google fonts Poppins提供のフォント。

app/templates

htmlファイルを入れるためのディレクトリです。

.gitignore

開発に関係なくリポジトリに上げる必要のないものを 除くためのファイルです。このファイルに必要ないものを記入しておけば、githubに上げる際に自動で除かれます。

requirements.txt

このファイルに開発に必要なライブラリを書いておけば、以下のコマンドで一括インストール可能です。

```
pip install -r requirements.txt
```

実行方法

app.pyがメインファイルとなります。以下のコマンドで、実行してください。

```
cd app
python app.py
```

ファイルごとの説明

以下で、各ファイルについて簡単に述べていきます。

app/models/Conference.py

会議室テーブルの定義をするファイル.

- コンストラクタ：Conference(name, capacity, fhoto_id, remarks)
- table name : conferences

- カラム
    - conference_id : 主キー，オートインクリメント（整数）
    - name : 会議室名（文字列，20文字上限）
    - capacity : 許容人数（整数）
    - fhoto_id : 写真番号（整数）
    - remarks : 備考（文字列，100文字上限）
- リレーション
    - reserves : 会議室と紐づく予約一覧（リスト）
    - equipments : 会議室と紐づく備品一覧（リスト）

app/models/ConferenceEquipment.py

会議室等と備品のテーブル（中間テーブル）を定義するファイル.

- table name : conference_equipments
- カラム
    - id : 備品・会議室id（整数，主キー，オートインクリメント）
    - conference_id : 会議室id（整数，外部キー）
    - equipment_id : 予約者id（整数，外部キー）
    - num : 備品の数（整数）

app/models/Equipment.py

備品テーブルの定義をするクラス

- table name : equipments
- カラム
    - equipment_id : 主キー，オートインクリメント（整数）
    - name : 備品名(文字列，30文字上限)
- リレーション
    - conferences : 複数の会議室と紐づく
- コンストラクタ：Equipment(name)

app/models/Register.py

登録者テーブルの定義をするクラス

- コンストラクタ：User(passwd, admin, user_id=None)
- table name : registers
- カラム
    - register_id : 主キー，オートインクリメント（整数）
    - user_id : ユーザーid（整数，外部キー）
    - passwd : パスワード(文字列，30文字上限)
    - admin : 管理者フラグ（Boolean）
- リレーション
    - reserves : 登録者と紐づく予約一覧（リスト）
    - users : 登録者と紐づく1名のユーザー

app/models/Reserve.py

予約テーブルの定義をするクラス

- コンストラクタ：Reserve(register_id, conference_id, date, starttime, endtime, purpose, remarks, user_id=None)
- table name : reserves
- カラム
    - reserve_id : 予約id（整数，主キー，オートインクリメント）
    - regiser_id : 予約者id（整数，外部キー）
    - conference_id : 会議室id（整数，外部キー）
    - date : 日付（文字列，30文字上限）
    - starttime : 予約開始時間（文字列，30文字上限）
    - starttime : 予約終了時間（文字列，30文字上限）
    - user_id : ユーザーid（整数，外部キー）
    - purpose : 目的（文字列，10文字上限）
    - remarks : 備考（文字列，100文字上限）
- リレーション
    - registers : 1名の予約者(Register)と紐づく
    - users : 1名の利用者(User)と紐づく
    - conferences : 1つの会議室と紐づく

app/models/User.py

ユーザーテーブルの定義をするクラス

- コンストラクタ：User(name, email)
- table name : users
- カラム
    - user_id : 主キー，オートインクリメント（整数）
    - name : ユーザーネーム(文字列，20文字上限)
    - email : メールアドレス(文字列，50文字上限)
- リレーション
    - reserves : ユーザーと紐づく予約一覧（リスト）
    - registers : ユーザーと紐づく1名の登録者(予約者)

app/models/database.py

データベースを初期化するファイル

- 変数
    - db
        - SQLAlchemyを通してデータベースを操作するための変数
- 関数
    - init_db(app)
        - アプリケーションの実体を受け取り、データベースの初期化、 マイグレーションの設定を行う.

        - app : Flaskアプリケーションの実体

app/models/database_test.py

データベースに関するテストを行うファイル

- 関数
    - add_user(name='豊橋太郎', email='tarou@example.com')
        - ユーザー追加のテスト用関数
        - 引数
            - name : ユーザー名（default='豊橋太郎'）
            - email : メールアドレス（default='tarou@example.com'）
        - 戻り値
            - なし
    - delete_user(email='tarou@example.com')
        - ユーザー削除のテスト用関数
        - 引数
            - email : メールアドレス（default='tarou@example.com'）
        - 戻り値
            - なし
    - update_user(old_email='tarou@example.com', new_email='toyohasi.tarou@example.com')
        - ユーザー情報更新のテスト用関数
        - 引数
            - old_email : 更新前メールアドレス（default='tarou@example.com'）
            - new_email : 更新後メールアドレス（default='toyohasi.tarou@example.com'）
        - 戻り値
            - なし
    - add_register()
        - 登録者追加のテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし
    - read_register()
        - 登録者情報読み込みのテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし
    - delete_register()

        - 登録者削除のテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - update_register()

        - 登録者情報更新のテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - add_equipment()

        - 備品追加のテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - read_equipment()

        - 備品情報読み込みのテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - delete_equipment(name='プロジェクター')

        - 備品削除のテスト用関数
        - 引数
            - name : 備品の名前
        - 戻り値
            - なし

    - update_equipment()

        - 備品情報更新のテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - add_conference()

        - 会議室追加のテスト用関数
        - 引数
            - なし

        - 戻り値
            - なし

- read_conference()

    - 会議室情報読み込みのテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- delete_conference()

    - 会議室削除のテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- update_conference()

    - 会議室情報更新のテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- add_reserve()

    - 予約追加のテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- read_reserve()

    - 予約情報読み込みのテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- delete_reserve()

    - 予約削除のテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- update_reserve()
    - 予約更新のテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし
- get_reserve()
    - 予約の日付重複に関するテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし
- test_User()
    - ユーザーテーブルの単体テスト（CRUD）用関数
    - 引数
        - なし
    - 戻り値
        - なし
- test_Register()
    - 登録者テーブルの単体テスト（CRUD）用関数
    - 引数
        - なし
    - 戻り値
        - なし
- test_Equipment()
    - 備品テーブルの単体テスト（CRUD）用関数
    - 引数
        - なし
    - 戻り値
        - なし
- test_Conference()
    - 会議室テーブルの単体テスト（CRUD）用関数
    - 引数
        - なし
    - 戻り値
        - なし
- test_Reserve()
    - 予約テーブルの単体テスト（CRUD）用関数

        - 引数
            - なし
        - 戻り値
            - なし

    - add_user_in_register()

        - 登録者とユーザー同時に登録できるかのテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - delete_user_and_register()

        - ユーザーを削除した際に登録者が削除されるかのテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - delete_register_and_user()

        - 登録者を削除した際にユーザーが削除されるかのテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - connect_conference_equipment()

        - 会議室と備品のむずびつけが行えるかのテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - delete_conference_without_equipment()

        - 会議室を削除した際に備品が削除されないかのテスト用関数
        - 引数
            - なし
        - 戻り値
            - なし

    - delete_equipment_without_conference()

        - 備品を削除した際に会議室が削除されないかのテスト用関数
        - 引数
            - なし
        - 戻り値

        - なし

- delete_reserve_without_conference_register_user()
    - 予約を削除した際に会議室，予約者，利用者が削除されないかのテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- delete_conference_and_reserve_without_register_user()
    - 会議室を削除した際に予約が削除されるかのテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- delete_register_and_reserve_without_conference_user()
    - 予約者を削除した際に予約が削除されるかのテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- delete_user_and_register_and_reserve_without_conference()
    - 利用者を削除した際に予約が削除されるかのテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

- all_unit_test()
    - 全てのテーブルの単体テストを行うためのテスト用関数
    - 引数
        - なし
    - 戻り値
        - なし

app/static/css/calendar_day.css

予約日表示カレンダーのデザインを設定するファイル

app/static/css/calenda_simple.css

予約簡易版表示のデザインを設定するファイル

app/static/css/calendar_week.css

予約週表示のデザインを設定するファイル

app/static/css/main.css

メイン画面のデザインを設定するファイル

app/static/css/main_admin.css

管理者画面のデザインを設定するファイル

app/static/css/main_reserve.css

予約画面のデザインを設定するファイル

app/static/css/main_tab.css

メイン画面のタブのデザインを設定するファイル

app/static/css/reserve.css

予約ページのボタンのデザインを設定するファイル

app/static/css/util.css

フォントサイズやパディングサイズ等の設定を記述しているファイル

app/static/js/calendar_day.js

- 予約の日表示の動作を記述するファイル
- 変数
    - today：現在の日時情報を保持する(Date)
    - showDate：表示する日時(Date)
    - day_disp：今日の日付表示用
- 関数
    - prev()
        - 前の日の予約を表示する関数
        - 引数
            - なし
        - 戻り値
            - なし
    - today_btn()
        - 当日の予約を表示する関数
        - 引数
            - なし
        - 戻り値
            - なし
    - next_btn()
        - 次の日の予約を表示する関数
        - 引数
            - なし

        - 戻り値
            - なし
    - showProcess(date)
        - 日付の表示を行う関数
        - 引数
            - date : 日付情報(Date)
        - 戻り値
            - なし
    - createProcess(year, month, dates)
        - カレンダーの作成，予約情報の表示，ポップアップ表示を行う関数
        - 引数
            - year : 表示する年
            - month : 表示する月
            - dates : 表示する日
        - 戻り値
            - calendar:カレンダー表示オブジェクト

app/static/js/calendar_week.js

- 予約の週表示の動作を記述するファイル
- 変数
    - week：曜日を格納する(リスト)
    - today：現在の日時情報を保持する(Date)
    - showDate：表示する日時(Date)
    - startDay：表示開始日時(文字列)
    - reserves：予約の一覧を保持する(リスト)
- 関数
    - prev()
        - 前の週の予約を表示する関数
        - 引数
            - なし
        - 戻り値
            - なし
    - current()
        - 今週の予約を表示する関数
        - 引数
            - なし
        - 戻り値
            - なし
    - next()
        - 次の週の予約を表示する関数
        - 引数
            - なし
        - 戻り値
            - なし
    - createCalendar()
        - カレンダーを作成する関数

        - 引数
            - なし
        - 戻り値
            - なし
    - createCalendar()
        - カレンダーの日にち表示部分を作成する関数
        - 引数
            - なし
        - 戻り値
            - calendar:カレンダー表示オブジェクト
    - createReserve()
        - カレンダーの予約表示部分，ポップアップ表示を作成する関数
        - 引数
            - なし
        - 戻り値
            - calendar:カレンダー表示オブジェクト
    - changeDate(day, num)
        - 日付の変更を行う関数
        - 引数
            - day : 変更前の日付
            - num : 進める日数
        - 戻り値
            - result : 'year/month/date'の形状の変更後の日付
    - setReserve(num)
        - 予約内容の取得を行う関数
        - 引数
            - num : 予約ID
        - 戻り値
            - なし
    - calendar_week_ajax()
        - 予約情報を取得するAjax通信を行う関数
        - 引数
            - なし
        - 戻り値
            - なし
    - calendar_ajax_GET()
        - 予約情報のGETを行う関数
        - 引数
            - なし
        - 戻り値
            - なし
    - calendar_ajax_POST(data)
        - 予約情報のPOSTを行う関数
        - 引数
            - data : 予約情報のデータ(object)
        - 戻り値

            - なし

app/static/js/main.js

- ログイン画面の設定を記述したファイル
- 関数
    - validate(input)
        - メールアドレス入力の判定を行う関数
        - 引数
            - input : 要素を特定するセレクタ文字列（String）
        - 戻り値
            - boolean : メールアドレスでない場合にfalseを返
    - showValidate(input)
        - アラート情報の表示を行う
        - 引数
            - なし
        - 戻り値
            - なし
    - hideValidate(input)
        - アラート情報の非表示（隠す）を行う
        - 引数
            - なし
        - 戻り値
            - なし

app/static/js/reserve.js

- 予約ページの動作を記述するファイル
- 変数
    - this_day：現在の日にち
    - this_month：現在の月
    - this_year：現在の年
    - today：今日の日付(Date)
    - max_year：年数の最大値，現在の年から数えて何年
- 関数
    - createListBox(start, end, id, def)
        - ListBoxの作成を行う関数
        - 引数
            - start : 最初に表示する値
            - end : 最後に表示する値
            - id : 表示部分のhtml要素のid
            - def : デフォルトの選択項目
        - 戻り値
            - 作成されたhtmlテキスト

app/templates/calendar/calendar_day.html

カレンダーの日表示を担うhtmlファイルです。

app/templates/calendar/calendar_simple.html

カレンダーの簡易表示を担うhtmlファイルです。

app/templates/calendar/calendar_week.html

カレンダーの週表示を担うhtmlファイルです。

app/templates/admin.html

管理者画面のhtmlファイルです。

app/templates/admin_eq.html

備品の追加・削除画面のhtmlファイルです。

app/templates/admin_room.html

会議室の追加・削除画面のhtmlファイルです。

app/templates/admin_user_add.html

管理者画面の中にある新規ユーザー追加画面のhtmlファイルです。

app/templates/error.html

何らかのエラーが発生した際に表示されるhtmlファイルです。

app/templates/index.html

ログイン画面のhtmlファイルです。

app/templates/main_admin.html

管理者画面における登録ユーザー情報の表示、削除、新規ユーザーや会議室登録への遷移、ログアウトボタンのhtmlファイルです。

app/templates/main_tab.html

ログインした際に表示されるメイン画面と、各機能への遷移ボタンについて書かれたhtmlファイルです。

app/templates/reserve.html

予約画面のhtmlファイルです。

app/templates/reserve1.html

予約画面の具体的な入力フォームについてのhtmlファイルです。

app/templates/room.html

会議室詳細についてのhtmlファイルです。

app/templates/template.html

各htmlファイルのひな型となるhtmlファイルです。

app/templates/test.html

機能のテストのためのものであり、実際のシステムには使用されていません。

app/admin_eq.py

admin_eq関数では、POSTメソッドを受け取った場合、更新であれば変数にその情報を格納し、更新します。エラーの場合はadmin_eq.htmlに飛ばします。 追加の場合は変数にその情報を格納し、追加します。エラーの場合はadmin_eq.htmlに飛ばします。 更新・追加の後はadmin_eq.htmlに飛ばします。

app/admin_main.py

admin_main関数では、最初にログインしているかの判定を行います。ログインしていなければログイン画面へリダイレクトします。ログインしていれば、Registerの内容をすべて取得しmain_admin.htmlへ情報を渡すと同時に飛ばします。

app/admin_room.py

admin_room関数では会議室情報をすべて取得しadmin_room.htmlへ情報を渡すと同時に飛ばします。
add_room関数では会議室情報をすべて取得しroom.htmlへ情報を渡すと同時に飛ばします。

app/admin_user_add.py

admin_user_add関数ではまずログインしているかの判定を行います。ログインしていなければログイン画面へリダイレクトします。 POSTメソッドが飛んでいた場合、フォームに入力されたemail、Name、passwordを取得します。 POSTメソッドでなかった場合は、admin_user_add.htmlへ飛ばします。 その後、adminにチェックが入っているかを判定して取得します。 もしすでにユーザとして登録されていた場合、Registerとしての情報があるか確認します。Regisiterとしての情報がなかった場合は、入力された情報を用いてRegisterとして情報を登録します。 ユーザとして登録されていなかった場合は、入力された情報をもとにユーザ、Registerとして情報を登録します。登録が終わったら、admin_main関数へリダイレクトします。

app/app.py

home関数では、ログイン画面へのリダイレクトを行っています。 main関数では、この予約システムの起動を行っています。

app/config.py

Flaskの実行設定ファイルです。

app/eq_delete.py

eq_delete関数は会議室の名前nameを引数として持っています。引数を用いて会議室を検索し、削除します。削除後はadmin_eq関数へリダイレクトします。

app/login.py

POSTメソッドを受け取った場合、フォームに入力されたemailとpasswordを取得します。そうでない場合はログイン画面へ飛ばします。 入力された情報をもとにRegisterとして登録されているか検索します。Registerとして登録されていてかつパスワードが正しい場合sessionを登録します。 adminとして登録されていた場合は管理者画面へ、そうでない場合はメイン画面へ飛ばします。 login_required関数はflagがsession中に存在し、かつsession["flag"]がTrueの場合ログインしていると判定してTrueを返します。そうでない場合はログインしていないと判定しFalseを返します。

app/main_tab.py

main_tab関数では、まずログインしているかの判定を行っています。ログインしていない場合はログイン画面に飛ばします。データベースからユーザ情報を取得し、格納します。その後、格納したリストとともにmain_tab.htmlへ飛ばします。 calendar_week関数ではまずログインしているかの判定を行います。ログインしていない場合はログイン画面に飛ばします。 POSTメソッドが飛んできた場合は予約IDを引数にreserve_delete関数を実行します。 データベースからユーザ情報、予約情報、会議室情報を取得し、格納します。そして、格納した情報を引数としてcalendar_week.htmlに飛ばします。 week_Ajax_GET関数では、まずログインしているかの判定を行います。ログインしていなければログイン画面へ飛ばします。POSTメソッドが飛んできた場合jsonファイルを返します。 week_Ajax_GET関数では、まずログインしているかの判定を行います。ログインしていなければログイン画面へ飛ばします。一週間を設定し、一週間分の予約を検索して格納します。その後それを返します。 calendar_day関数では、まずログインしているかの判定を行います。ログインしていなければログイン画面へ飛ばします。 リストに、予約情報と会議室情報をすべて格納し、それを引数としてcalendar_day.htmlへ飛ばします。 calendar_simple関数では、まずログインしているかの判定を行っています。ログインしていない場合はログイン画面に飛ばします。予約情報を取得してリストに入れた後、ソートします。会議室情報も取得してリストに入れます。その後、それらを引数にcalendar_simple.htmlへ飛ばします。 reserve_page関数では、reserves関数へのリダイレクトを行っています。 room_page関数では、会議室情報を取得し、それを引数としてroom.htmlへ飛ばします。 logout関数では、まずログインしているかの判定を行っています。ログインしていない場合はログイン画面に飛ばします。その後、セッション情報を削除し、ログイン画面へリダイレクトします。 reserve_delete関数では、管理者かその予約をした人のみ予約を削除できるという機能を実装しています。 reserve_list関数では予約情報をリストにし、返しています。

app/reserves.py

reserves関数では、まずログインしているかの判定を行っています。ログインしていない場合はログイン画面に飛ばします。 POSTメソッドを受け取った場合、まず入力フォームから予約開始時間と終了時間を取得します。 予約開始時間より終了時間が早かったら会議室情報を取得しreserve.htmlに飛ばします。 そうでない場合はRegister情報と残りのフォームからの情報を取得します。 予約の排他制御に関する判定を行い、それらをクリアした場合は予約登録を行います。 予約登録後は予約画面へ飛ばします。 予約情報にエラーがあって登録できなかった場合は、errorがあるという情報を持ったうえで予約画面へ飛ばします。

app/room_delete.py

room_delete関数は会議室の名前nameを引数として持っています。引数を用いて会議室と備品の検索をし、備品を削除します。次に会議室の画像ファイルがあった場合それを削除します。最後に会議室情報を削除し、admin_room関数へリダイレクトします。

app/test.db

データベースファイルです。

app/user_delete.py

user_delete関数は、userのemail情報を引数に持っています。 関数では、まずログインしているかの判定を行います。ログインしていない場合はログイン画面へ飛ばします。 引数user_emailが現在ログインしているユーザでない場合、ユーザの削除を行います。削除後はadmin_main関数へリダイレクトします。

データベースに関して

データベースに用いたシステム・パッケージ

- SQLite
- SQL Alchemy
- Flask Migrate

データベース定義

データベースのテーブルとそれらの関係について示す.

テーブル関係図

テーブル間のリレーションに関しては以下の図を参照していただきたい.

関係図

(参照/database_ver2.pdf)

User

ユーザーテーブルの定義

- table name : users
- カラム
  - user_id : 主キー，オートインクリメント（整数）
  - name : ユーザーネーム(文字列，20文字上限)
  - email : メールアドレス(文字列，50文字上限)
- リレーション
  - reserves : ユーザーと紐づく予約一覧（リスト）
  - registers : ユーザーと紐づく1名の登録者(予約者)
- コンストラクタ：User(name, email)

Register

登録者テーブルの定義

- table name : registers
- カラム
  - register_id : 主キー，オートインクリメント（整数）
  - user_id : ユーザーid（整数，外部キー）
  - passwd : パスワード(文字列，30文字上限)
  - admin : 管理者フラグ（Boolean）

- リレーション
    - reserves : 登録者と紐づく予約一覧（リスト）
    - users : 登録者と紐づく1名のユーザー
- コンストラクタ：User(passwd, admin, user_id=None)

Conference

会議室テーブルの定義

- table name : conferences
- カラム
    - conference_id : 主キー，オートインクリメント（整数）
    - name : 会議室名（文字列，20文字上限）
    - capacity : 許容人数（整数）
    - fhoto_id : 写真番号（整数）
    - remarks : 備考（文字列，100文字上限）
- リレーション
    - reserves : 会議室と紐づく予約一覧（リスト）
    - equipments : 会議室と紐づく備品一覧（リスト）
- コンストラクタ：Conference(name, capacity, fhoto_id, remarks)

Reserve

予約テーブルの定義

- table name : reserves
- カラム
    - reserve_id : 予約id（整数，主キー，オートインクリメント）
    - regiser_id : 予約者id（整数，外部キー）
    - conference_id : 会議室id（整数，外部キー）
    - date : 日付（文字列，30文字上限）
    - starttime : 予約開始時間（文字列，30文字上限）
    - starttime : 予約終了時間（文字列，30文字上限）
    - user_id : ユーザーid（整数，外部キー）
    - purpose : 目的（文字列，10文字上限）
    - remarks : 備考（文字列，100文字上限）
- リレーション
    - registers : 1名の予約者(Register)と紐づく
    - users : 1名の利用者(User)と紐づく
    - conferences : 1つの会議室と紐づく
- コンストラクタ：Reserve(register_id, conference_id, date, starttime, endtime, purpose, remarks, user_id=None)

Equipment

備品テーブルの定義

- table name : equipments
- カラム

    - equipment_id : 主キー，オートインクリメント（整数）
    - name : 備品名(文字列，30文字上限)
- リレーション
    - conferences : 複数の会議室と紐づく
- コンストラクタ：Equipment(name)

ConferenceEquipment

このテーブルは中間テーブルの役割なので、基本的には操作しない． 会議室等から備品数を取得する際にnumを参照する．

- table name : conference_equipments
- カラム
    - id : 備品・会議室id（整数，主キー，オートインクリメント）
    - conference_id : 会議室id（整数，外部キー）
    - equipment_id : 予約者id（整数，外部キー）
    - num : 備品の数（整数）

データベース（SQL）の使い方

ここでは、基本的なCRUD操作について記述する． 詳細については、SQLAlchemyや基本的な使い方を参照していただきたい． また、例としてユーザーテーブルの操作を挙げているが、他のテーブル名に置き換えれば動作する．

Create （データの追加）

- テーブル単体での追加

```
# データベースインスタンスを読み込む
from models.database import db

# ユーザーをインスタンス化
user = User(name='tarou', email='tarou@example.com')

# 追加する
db.session.add(user)
# 仮にデータベースに反映させる（エラー処理等に使える）
db.session.flush()
# 実際にデータベースに反映させる
db.session.commit()
```

以下に，リレーションを用いた便利な方法を記述する．

- リレーションと同時にユーザー登録を行う（リレーションが1名の場合）

  ユーザーをリレーションに追加すれば自動的にユーザーテーブルにも追加される．

```
# 登録者を作成する
register = Register('passwd', False)
```

```
# リレーションにユーザーを追加
register.users = User('豊橋花子', 'hanako@example.com')
db.session.add(register)
db.session.flush()
db.session.commit()
# 自動的にUserテーブルに豊橋花子が追加される
```

- リレーションと同時に備品登録を行う（リレーションが複数の場合）

  リレーションが複数の場合（仕様のリスト表記があるもの）はリストとして扱う

```
conference = Conference('会議室A', 60, 1, '特になし')
conference.equipments.append(Equipment('プロジェクター'))
db.session.add(conference)
db.session.flush()
db.session.commit()
```

Read (データの読み出し)

- リレーションなし

```
# 全てのユーザーの読み込み
users = User.query.all()
# 名前でフィルターを掛ける(1人の)場合
user = db.session.query(User).filter_by(name='tarou').first()
# 変数名.キーで中身の値を取り出せる.（キーは各テーブルの仕様参照）
print(user.name)
```

- リレーションあり

```
# 予約の取得
reserve = db.session.query(Reserve).filter(Reserve.id==1).first()
# リレーションを張ったユーザー（1名）が取得できる
user = reserve.users
# リレーションを張った会議（1部屋）が取得できる
conference = reserve.conferences
```

- 特殊事例：中間テーブルの情報を取り出す方法

  参考の中間テーブルに カラムがある場合の方法を参考にしてデータを取得してください.

Delete（データの削除）

- リレーション無し

```

```

```
# ユーザーの読み出し(read)
user = db.session.query(User).filter_by(name='tarou').first()

# 削除する
db.session.delete(user)
# 仮にデータベースに反映させる（エラー処理等に使える）
db.session.flush()
# 実際にデータベースに反映させる
db.session.commit()
```

- リレーションあり

  リレーションなしの場合と削除方法は変わらないが、 以下の関係性が成り立っていることに注意してもらいたい.

- UserとRegister
  - Userを削除した場合Registerも削除される
  - Registerを削除した場合Userは削除されない
- ConferenceとEquipment
  - 会議室を削除した際に備品は削除されない
  - 備品を削除した際に会議室は削除されない
- Reserveと(User, Register, Conference)
  - 予約を削除した際に会議室，予約者，利用者を削除しない
  - 会議室を削除した際に予約を削除する(予約者，利用者は削除されない)
  - 予約者を削除した際に予約を削除する(会議室，利用者は削除されない)
  - 利用者を削除した際に予約を削除する(会議室，予約者は削除されない)
  - ただし、利用者と予約者が同一だった場合は、UserとRegisterの関係により処理される

Update（データの更新）

```
# ユーザーの読み出し(read)
user = db.session.query(User).filter_by(name='tarou').first()
# メールアドレスの変更
user.email = 'tarou@example.com'

# 更新する
db.session.add(user)
# 仮にデータベースに反映させる（エラー処理等に使える）
db.session.flush()
# 実際にデータベースに反映させる
db.session.commit()
```

特定の情報を更新すれば、リレーション先の情報も自動的に書き換わる

データベースの確認方法

起動

sqliteコマンドでデータベースを開く.

```
sqlite3 test.db
```

終了

```
sqlite > .quit
```

テーブル一覧

テーブル一覧を表示

```
sqlite > .table
```

テーブルの中身取得

テーブルからデータを取得して表示するSQL文.

usersを任意のテーブル名にすることで表示可能.

```
sqlite > select * from users;
```

テーブルの要素一覧

テーブル要素についての情報が見られる.

```
sqlite > .schema
```

メンテナンス用

1. migrateを行ってデータベース更新
2. flask shellを使ったテスト

Migrateの手順

1. migration設定の初期化 （初回のみ）

```
flask db init
```

2. 変更する・追加するテーブルファイルをmodels内で編集・作成
3. database.pyにてテーブルクラスをimportする
4. migrationファイルの作成

   前回との差分をmigrationファイルとして生成.

   ```
   flask db migrate
   ```

5. migrationの実行

   migrationファイルを実行する.（実際の書き換え）

   ```
   flask db upgrade
   ```

※バージョンを戻す（rollback）

```
flask db downgrade
```

※要注意：データベースを最初に戻す

```
flask db upgrade head
```

運用サーバーへの移行に関して

本プロジェクトでは、運用サーバーへのデプロイを行えなかったため、 以下に運用サーバーへのデプロイを想定した操作を記述する.

ローカルサーバーと運用サーバーでの変更予定点

ローカルサーバーと運用サーバーでの変更予定点を以下に記述する.

シボレス認証の導入

- 学内サーバーに設置する場合、シボレス認証を導入する必要がある.
- 現行のシステムのログイン画面をシボレスのログイン画面に変更する必要がある.
- ログイン画面で保持しているセッション情報をシボレスから取得する必要がある.

データベース管理システムの変更

- 学内サーバーに設置する場合、データベースをMySQLに変更する必要がある.
- config.py内にあるデータベースの設定の記述をMySQL用に書き換える必要がある.
- その他、テーブル等の変更は必要ない

cgi機能による実行

- ローカル環境では、Flaskの機能のみで動作を行っている.
- 学内サーバーでに設置する場合、FastCGIを用いる必要がある.

サーバーへの移行方法

学内サーバーへの公開手順を以下に示す.

1. ホスティングサーバーの利用申請をする.
2. ホスティングサーバーの利用手順を参照し、シボレス認証の設定を行う.
3. 必要なパッケージをユーザーにインストールする. （必要なパッケージはrequirements.txtに記述済み）
4. 上記に記述した成果物の変更点を開発サーバー用に変更する.
5. 成果物をサーバーに設置する.
6. データベースの初期化(migration)を行う.
7. システムが動作していることを確認する.