DB

① UI とレスポンシブデザイン

② 権限管理(削除権限)

＜解決策＞

id を工夫する。管理者を 0 とし、新入生は 1 からスタートするようにする。また、初登録の際に管理者か新入生、どちらとして登録するかを選択する。

＜実装方法＞

1. 管理者の Id カラムを integer から string に変更
2. 管理者 ID が 001 から始まるように設定、例として管理者 ID を 3 桁にしているため、1000 人の管理者が登録可能

```
def generate_admin_id():
 # 既存の管理者 ID を取得
    last_admin=
Members.query.filter(Members.id.like('0%')).order_by(Members.id.desc()).first()

    if last_admin:
        # 最後の管理者 ID を基に次の ID を生成
        last_id = int(last_admin.id)
        new_id = f"{last_id + 1:03d}"  # 3 桁にフォーマット
    else:
        new_id = "001"  # 初回登録時は "001"

    return new_id
```

3. 生成した管理者 ID を登録する

```
@app.route('/init_admin', methods=['POST'])
def init_admin():
    # 管理者 ID を生成
    admin_id = generate_admin_id()

    # 管理者を登録
    admin = Members(
        id=admin_id,  # 管理者 ID
        name='Admin',
        grade=0,  # 管理者専用の値
        club_id=1,  # 管理者専用クラブ ID
        contact='admin@example.com'
    )
    db.session.add(admin)
    db.session.commit()

    return jsonify({'message': f'Admin created with id={admin_id}'})
```

4. 新入生の ID は最新の新入生の ID の次の値を生成する

```
def generate_member_id():
    # 管理者以外の最大 ID を取得（1 以上の ID のみ）
    last_member = Members.query.filter(Members.id >=
1).order_by(Members.id.desc()).first()
    if last_member:
        return last_member.id + 1
    else:
        return 1  # 最初の新入生は ID=1
```

5.　生成した新入生 ID を登録する

```
@app.route('/add_member', methods=['POST'])
def add_member():
    name = request.form['name']
    grade = int(request.form['grade'])
    club_id = int(request.form['club_id'])
    contact = request.form['contact']

    # ID を自動生成
    member_id = generate_member_id()

    new_member = Members(
        id=member_id,  # 自動生成された ID を設定
        name=name,
        grade=grade,
        club_id=club_id,
        contact=contact
    )
    db.session.add(new_member)
    db.session.commit()

    return redirect(url_for('home'))
```

③ エラー処理

＜解決策＞

入力データを検証し、データ型や値のズレがないかを確認する

＜実装方法＞

新入生、管理者、部活の登録時にデータ検証を行う、エラーが発生した場合、それに応じたメッセージを戻り値として返す

1. 新入生登録時のエラー処理

```
@app.route('/add_member', methods=['POST'])
def add_member():
    try:
        # 入力値の取得
        name = request.form.get('name')
        grade = request.form.get('grade', type=int)
        club_id = request.form.get('club_id', type=int)
        contact = request.form.get('contact')

        # 必須フィールドのチェック
        if not name or not grade or not club_id or not contact:
            return jsonify({'error': 'All fields are required'}), 400

        # 値の検証
        if grade < 1 or grade > 4:
            return jsonify({'error': 'Grade must be between 1 and 4'}), 400

        if not isinstance(club_id, int):
            return jsonify({'error': 'Invalid club ID'}), 400

        # 新しいメンバーの作成
        new_member = Members(
            name=name,
            grade=grade,
            club_id=club_id,
            contact=contact
        )
```

```
            db.session.add(new_member)
            db.session.commit()

            return jsonify({'message': 'Member added successfully'}), 201

        except Exception as e:
            return jsonify({'error': 'An unexpected error occurred', 'details': str(e)}), 500
```

2.　管理者登録時のエラー処理
名前、連絡先、学年がからでないこと、ID が 0 から始まる三桁の数字であること、
管理者の ID が重複しないことを確認する。

```
@app.route('/add_admin', methods=['POST'])
def add_admin():
    try:
        # 入力データの取得
        name = request.form.get('name')
        contact = request.form.get('contact')
        grade = request.form.get('grade', type=int)
        admin_id = request.form.get('id', type=int)

        # 1. 必須フィールドの検証
        if not name or not contact or grade is None or admin_id is None:
            return jsonify({'error': 'All fields are required'}), 400

        # 2. ID の検証（0 から始まる 3 桁以上の数字）
        if not str(admin_id).startswith('0') or len(str(admin_id)) < 3:
            return jsonify({'error': 'Admin ID must start with 0 and be at least 3 digits
long'}), 400

        # 3. 既存管理者の確認
        existing_admin = Members.query.filter(Members.id.like('0%')).first()
        if existing_admin:
            return jsonify({'error': 'An admin already exists'}), 400
```

```
        # 4. グレードの検証
        if grade < 1 or grade > 4:
            return jsonify({'error': 'Grade must be between 1 and 4'}), 400

        # 5. 管理者の登録
        new_admin = Members(
            id=admin_id,
            name=name,
            grade=grade,
            club_id=None,  # 管理者は特定のクラブに属さない
            contact=contact
        )
        db.session.add(new_admin)
        db.session.commit()

        return jsonify({'message': 'Admin registered successfully!'}), 201

    except Exception as e:
        return jsonify({'error': 'Failed to register admin', 'details': str(e)}), 500
```

3. 部活登録時のエラー処理

```
@app.route('/clubs', methods=['POST'])
def create_club():
    try:
        data = request.json

        # 必須フィールドのチェック
        if not data.get('club_name') or not data.get('capacity'):
            return jsonify({'error': 'club_name and capacity are required'}), 400

        # 値の検証
        if data['capacity'] <= 0:
            return jsonify({'error': 'Capacity must be greater than 0'}), 400

        # 部活の登録
```

```
            new_club = Clubs(
                club_name=data['club_name'],
                description=data.get('description', ''),
                capacity=data['capacity']
            )
            db.session.add(new_club)
            db.session.commit()

            return jsonify({'message': 'Club created successfully'}), 201

        except Exception as e:
            return jsonify({'error': 'Failed to create club', 'details': str(e)}), 500
```

④ 部活の詳細確認機能

＜解決策＞/＜実装方法＞

部活の定員と現在の登録人数を返すコードを追加する

```
@app.route('/clubs/<int:id>', methods=['GET'])
def get_club_details(id):
    club = Clubs.query.get(id)
    if not club:
        return jsonify({'error': 'Club not found'}), 404

    return jsonify({
        'id': club.id,
        'club_name': club.club_name,
        'description': club.description,
        'capacity': club.capacity,
        'current_members': club.current_members
    }), 200
```

⑤ テーブル設計の拡張

＜解決策＞

Club テーブルに現在の参加人数を記録するカラムを新たに作成し、動的に管理する

＜実装方法＞

1. current_members をついかする

```
class Clubs(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    club_name = db.Column(db.String(50), nullable=False)
    description = db.Column(db.Text, nullable=True)
    capacity = db.Column(db.Integer, nullable=False)  # 部活の定員
    current_members = db.Column(db.Integer, nullable=False, default=0)  # 現在の登録人数
```

2. 新入生登録時の人数更新のためのコードを追加する、capacity を超えたときエラー

```
@app.route('/add_member', methods=['POST'])
def add_member():
    try:
        name = request.form['name']
        grade = int(request.form['grade'])
        club_id = int(request.form['club_id'])
        contact = request.form['contact']

        # 対象の部活を取得
        club = Clubs.query.get(club_id)
        if not club:
            return jsonify({'error': 'Club not found'}), 404

        # 定員チェック
        if club.current_members >= club.capacity:
            return jsonify({'error': 'Club is already at full capacity'}), 400

        # メンバー登録
        new_member = Members(
            name=name,
```

```
                grade=grade,
                club_id=club_id,
                contact=contact
            )
            db.session.add(new_member)

            # 現在のメンバー数を更新
            club.current_members += 1
            db.session.commit()

            return jsonify({'message': 'Member added successfully!'}), 201

        except Exception as e:
            return jsonify({'error': 'Failed to add member', 'details': str(e)}), 500
```

3.メンバーが削除されたときの人数更新するコードを追加

```
  @app.route('/members/<int:id>', methods=['DELETE'])
def delete_member(id):
    try:
        member = Members.query.get(id)
        if not member:
            return jsonify({'error': 'Member not found'}), 404

        # 部活の現在メンバー数を減らす
        club = Clubs.query.get(member.club_id)
        if club:
            club.current_members = max(club.current_members - 1, 0)

        db.session.delete(member)
        db.session.commit()
        return jsonify({'message': 'Member deleted successfully!'}), 200

    except Exception as e:
        return jsonify({'error': 'Failed to delete member', 'details': str(e)}), 500
```