卒業論文

研究室用グループウェアアプリの開発

関西学院大学理工学部

情報科学科 西谷研究室

27020731 細井 大輝

2024 年 3 月

概 要

西谷研究室では情報共有ツールとして Discord を使用している [1]．しかし，Discord は過去に共有されたデータが整理されずグループメンバーが混乱してしまい，継続的な利用が困難である．本研究では，研究室内の情報共有ツールとしてデータを参照する際の混乱を避け，データを共有・見返す作業を習慣づけることを目的としたグループウェアアプリ Habit man の開発を行った．データを整理する方法として「超」整理法 [2] と呼ばれる手法を応用して共有データを一括管理する．主なアプリの特徴として，日付をキーとしたデータの保存，非アクティブデータを別のテーブルへと移行するアーカイブ機能，データの絞り込みを容易にする為のタグ，キーワード検索機能，リアルタイムな情報提供を可能にする通知機能がある．結果として，必要なデータの検索と管理が容易になり，グループメンバーの継続的な利用に繋げることができた．

目 次

図 目 次

第1章　序論

西谷研究室では情報共有ツールとしてDiscordを使用していた．しかし，Discordは「オンラインゲームをしながら、世界中の友達とコミュニケーションをとる」という課題を解決するために開発されたという背景があるため，研究室用の情報共有ツールとしては使用しづらい点がいくつかある．

1つ目は，投稿されたデータは降順に表示されるため，過去に共有されたデータはスクロールして遡るしかない点である．Discordでは共有されたデータの分類も行われず，ただ最新の共有データが1番下部に表示されるシステムであるため，データを見返す作業を困難にしてしまう．2つ目は，データのタグ付け機能が存在しないため，データの絞り込みが困難な点である．人間は過去に情報共有されたデータを見返す際に内容で検索するよりタグなどの内容に紐付けられた単語で検索する方が簡単であるため，タグ付け機能は必須であるが，Discordには存在しない．3つ目は，日付ごとのデータ管理ができないため，整理と見返しが困難な点である．Discordでは日付をキーとしてデータを絞り込むことはできない．自分自身が必要なデータは新しいものの場合が多く，日付というキーはその必要なデータを高速に検索するために必須のキーである．これらの難点は共有されたデータが整理されずグループメンバーが混乱してしまい，継続的に利用されなくなるという可能性を生んでしまう．

Discord以外にも，Slackという情報共有ツールが存在する[3]．こちらのツールは「人々をそれぞれが必要とする情報につなげる、ビジネス用のメッセージングアプリ」という開発背景があるが，使用感はDiscordとほぼ同様であり，難点も似ている．

よって，本研究ではデータを参照する際の混乱を避け，データを共有・見返す作業を習慣づけることを目的としたグループウェアアプリ Habit man を開発した．

第2章　手法

2.1　開発手法

設計・実装・デプロイを短期間に繰り返して研究室内メンバーから得た価値を学習し適応するためアジャイル開発手法を採用した [4]．アジャイル開発は人間・迅速さ・顧客・適応性に価値を置くソフトウェア開発手法であり，ユーザーから得た反応から方向性や仮説を見出し，ユーザーへ価値を素早く届け，実戦投入の学びから素早く改善を行うというサイクルを確立することができる．

2.1.1　Ruby on Rails

開発環境として MIT ライセンスに基づいて Ruby で書かれたサーバー側 Web アプリケーションフレームワークである Ruby on Rails(Rails) を選定した [5]．Rails を選定した理由は 3 つある．

最初の理由は広いプラットフォームからのアプリ利用が可能な点である．Rails を利用したソフトウェアは Web ブラウザ上で動作する為，PC のみならず，スマートフォンやタブレットなどの様々なデバイス上でアプリケーションを利用することができる．一方で，デスクトップアプリケーションやネイティブアプリケーションは利用するために一旦，アプリケーションのダウンロードを行う必要がある．

次の理由は日本語のドキュメントが充実しているため開発ハードルが低い点である．Ruby は日本人であるまつもとゆきひろ氏 [6] によって作られたプログラミング言語であるため，他の言語より日本語のドキュメントが充実している．よって，ドキュメントを日本語に訳す手間やコストを大幅に削減しながら開発に取り組むことができる．また，Ruby を使っているエンジニアも多く Qiita[7] などの主にエンジニアやプログラマーを対象とした，技術的な知識を共有し合うためのプラットフォーム上での文献も充実しているため，

実装に悩む時間なども削減することができる．

最後にライブラリが豊富な点である．Ruby には RubyGems(gem) と呼ばれるパッケージマネージャーが用意されており，多くのライブラリが配布されている [8]．gem によって，複雑な実装が簡素化され，短時間で Web アプリケーションの開発，拡張が可能になる．これは，アジャイル開発を行う上で必要な要素になる．

2.1.2 Heroku

Web アプリの運用プラットフォームとして Heroku を選定した [9]．Heroku とは，クラウドプラットフォームとして提供される Platform as a Service(PaaS) サービスで，開発者がアプリケーションを開発，デプロイ，運用するための簡単で柔軟な環境を提供している．Heroku はコマンドラインツール以外にも Web インターフェースを使用して，アプリケーションのコードや依存関係をアップロードし，自動的にデプロイのプロセスを処理する．加えて，Add-ons[10] と呼ばれるデータベース，キャッシュ，ログ分析，モニタリングなど，様々なサードパーティのサービスを利用することが可能である．Add-ons により，容易にアップロード画像の管理，ユーザーへのメール送信などを実装することができるようになる．また，Heroku と Rails は同じ哲学と原則に基づいているため，組み合わせが最適である．例として Rails は「Convention over Configuration」という原則に基づいている [11]．これは，開発者の決定すべきことを減少させ，単純にするが柔軟性は失わせないソフトウェア設計パラダイムのことである．Heroku も同様に，多くの設定や構成を自動化し，開発者が最初から効率的にアプリケーションをデプロイできるように設計されている．Heroku は PostgreSQL をデフォルトのデータベースとして提供しており，Rails も PostgreSQL を推奨している．

第3章　結果と考察

開発したWebアプリはHerokuに公開している [12]．Habit manはRailsの標準ライブラリを使用しているため現バージョン (7.0.4) がサポートする．

3.1　アプリ機能

今回，開発したアプリでは大きく分けて5つの機能を実装した．それぞれの機能は図3.1の通りである．

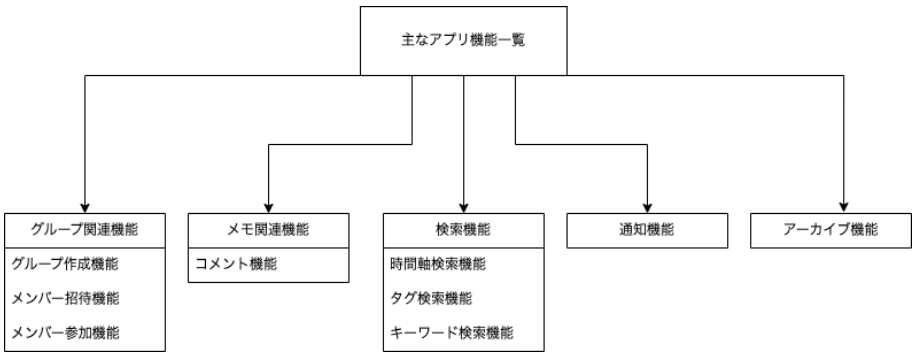

図 3.1: アプリ機能一覧.

3.1.1　グループ関連機能

1つ目はグループ関連機能であり，グループウェアアプリを開発していく上で，グループ作成機能，メンバー招待機能，メンバー参加機能の3つの機能が必須となる．グループ一覧画面からボタンを押下すると，グループを作成するためのモーダル画面が表示され，任意のグループ名を定めると作成することができる．グループメンバーを招待する際は，メンバー一覧画面からボタンを押下すると，招待したいメンバーのメールアドレスを入力するためのモーダル画面が表示され，アプリに登録済みのユーザーを招待することができ

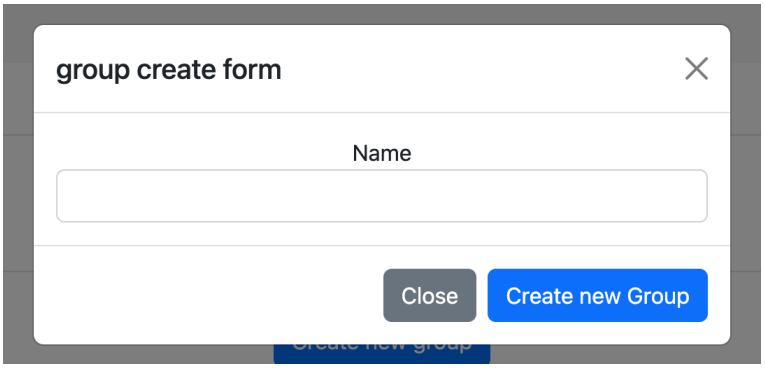

図 3.2: グループを生成する動作.

る．アプリに登録していないユーザーのメールアドレスを入力するとエラーが表示される．招待されたメンバーがグループに参加するためには招待メールの「Join the group」

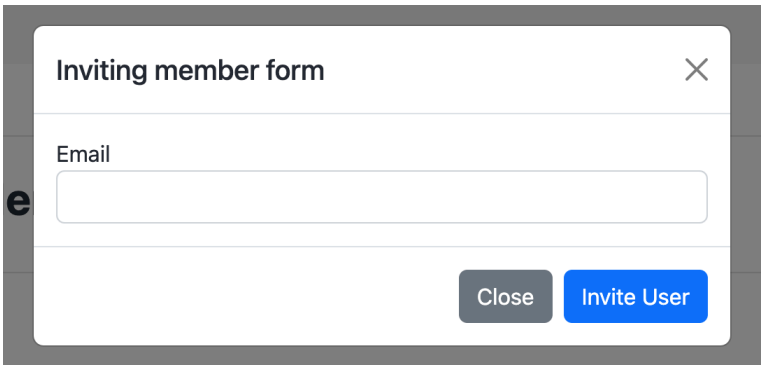

図 3.3: グループメンバーを招待する動作.

というリンクを押下する必要がある．

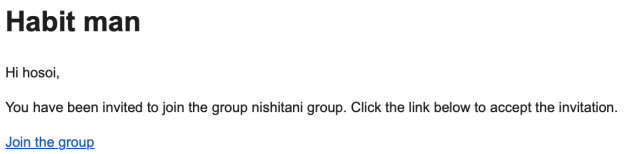

図 3.4: 招待されたグループに参加する動作.

3.1.2 メモ関連機能

2つ目はメモ関連機能であり，グループ内で情報を共有するためにCreate(生成)，Read(読み取り)，Update(更新)，Delete(削除)(CRUD) 処理は必須である．メモを生成する際は図

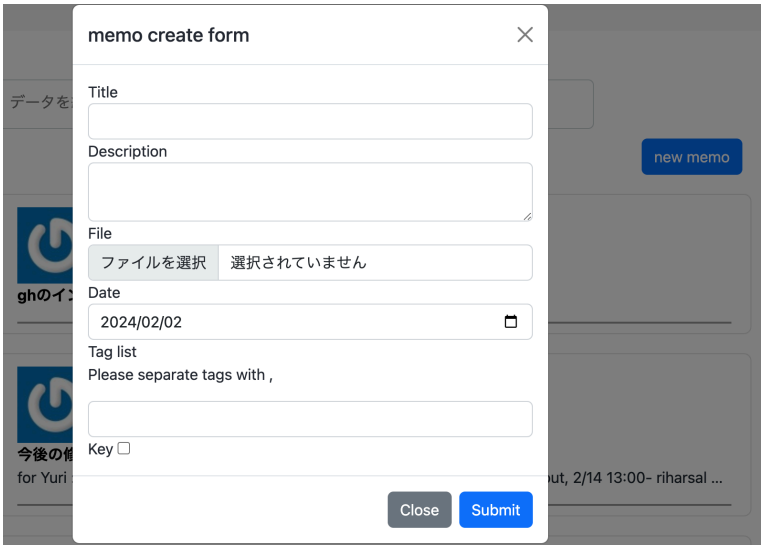

図 3.5: メモを生成する動作.

3.5 のように日付を選択する必要があるため，必然的に日付がキーとして保存される．日付をキーとして情報を保存するメリットは以下の 2 点である．

1. 投稿されたデータは保存したい日付を 1 つのカラムに保持しているため，日付ごとのデータ管理が容易.

2. データをカテゴリごとに分類する必要がないため，その他といった曖昧なデータ分類が無くなる.

また，投稿するメモは key という真偽値型のカラムを保持している．key カラムが真の場合は付録 A.5 に述べるアーカイブに移動しない．この key カラムの値はメモの編集画面にて変更することが可能である.

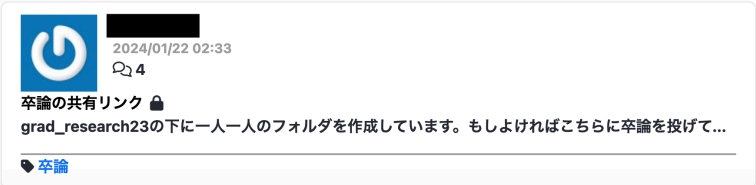

図 3.6: 鍵付きメモ.

3.1.3　検索機能

3つ目は検索機能であり，グループ内で共有されたデータを参照するためには検索機能は欠かせない．今回，開発したWebアプリでは主に日付をキーとして検索する時間軸検索，共有されたデータの目印をキーとして検索するタグ検索，タイトルの語句をキーとして検索するキーワード検索に分けて検索機能を実装した．

3.1.4　アーカイブ機能

4つ目はアーカイブ機能であり，今回開発したような情報共有アプリにアーカイブ機能を搭載するメリットは以下の5点である．

1. データの整理と保持．
2. Webアプリ全体のパフォーマンス向上．
3. 検索の向上．
4. データの復元と復旧．
5. ストレージの最適化．

まず1.に関して，長期間にわたって蓄積されるデータは，アクセスが減少するか，またはほとんど必要ない場合がある．アーカイブ機能を使用することで，これらのデータを整理し，必要なときに簡単にアクセスできるようになる．次に2.に関して，大量のデータがアクティブなデータとしてデータベースに保持されると，データベースへのアクセス速度が低下する可能性がある．アクティブなデータとアーカイブデータを分離することで，アプリケーションのパフォーマンスを向上させることができる．次に3.に関して，アクティブなデータセットが小さくなると，検索やクエリの速度が向上する．アーカイブ機能を使用してアクティブなデータとアーカイブされたデータを分離することで，特定のデータを検索する際に処理が迅速になる．次に4.に関して，アクセスが減少したデータを削除するのではなく，アーカイブにデータをセットすることによってデータの復元をする必要がなくなる．最後に5.に関して，アクティブなデータとアーカイブデータを分離することで，データベースやストレージの使用量を最適化できる．アクティブなデータに対し

ては高速なストレージを使用し，アーカイブデータはコストのかかる高性能なストレージに移動することが可能である．

3.1.5　通知機能

5つ目は通知機能であり，今回開発したような情報共有アプリに通知機能を搭載するメリットは以下の4点である．

1. リアルタイムな情報提供.
2. エンゲージメント向上.
3. ユーザーエクスペリエンスの向上.
4. 重要な情報の警告.

まず1.に関して，通知機能により，ユーザーはリアルタイムで新しい情報や重要なイベントにアクセスできる．例えば，新しいメッセージ，更新されたデータ，または特定のアクションが実行されたことを通知できる．次に2.に関して，ユーザーに対して新しい情報やアクションを通知することで，アプリへのエンゲージメントが向上する．通知はユーザーの注意を引きつけ，アプリの利用頻度を増加させる効果がある．次に3.に関して，通知はユーザーエクスペリエンスを向上させる．ユーザーはアプリを開かなくても新しい情報を受け取ることができ，アプリの使用が便利になる．最後に4.に関して，出席連絡や期限に関する通知は，重要事項であるため，高確率でユーザーに行き届かないと意味がない．通知機能はユーザーにこれらの情報を素早く，確実に伝達することができる．Habit manはメールでの通知を基本とする．メール通知はCreate，Update処理をした際にグループメンバー全員に行き届くようにした．

3.2　日付軸管理

投稿されたデータは保存したい日付を1つのカラムに保持しているため，日付ごとのデータ管理が容易となった．また，投稿されたデータ一覧画面では，タグやキーワードでのデータ絞り込み機能が利用可能である．また，どちらのアプリもログイン済みであるこ

とを前提として，投稿されたデータを参照するまでのアクション数を比較する．Discord ではサーバーの選択，チャンネルの選択，画面のスクロールなど最低でも 3 つのアクションを要する．Habit man では日付選択の 1 アクションでデータを参照することが可能である．

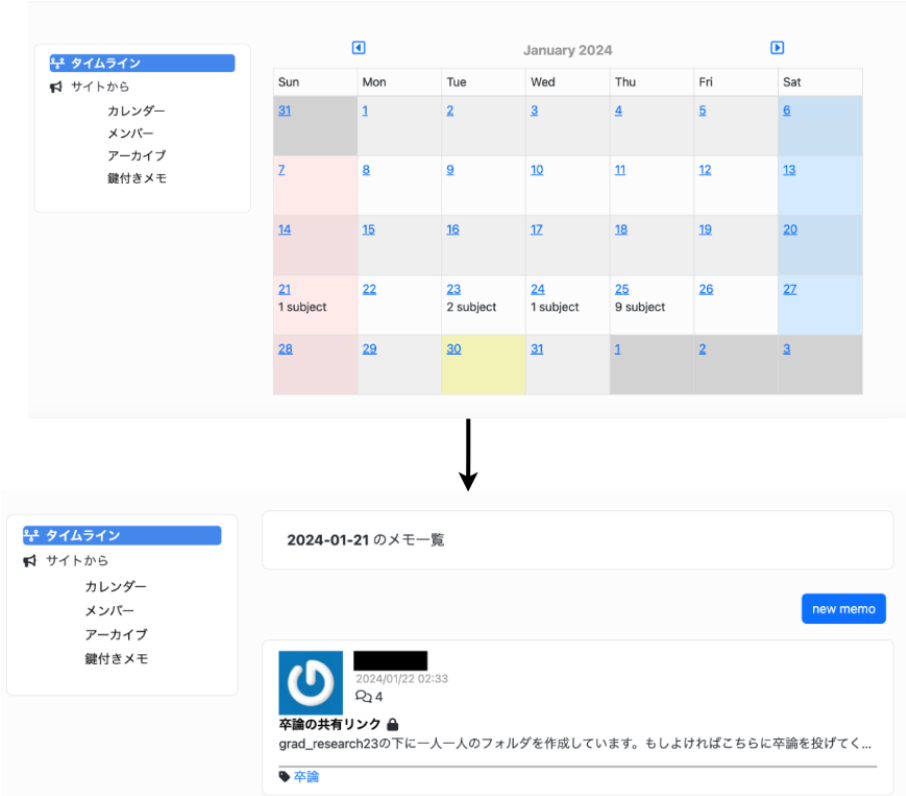

図 3.7: 参照したい日付に保存されたデータを参照する一連の動作.

3.3 ローカル環境から本番環境へ

3.3.1 Heroku × Github でのデプロイ方法

CLI を使用した Heroku への Web アプリケーションのデプロイ方法が存在するが，今回は Github 連携を使用したデプロイ方法を記す．Heroku は Github と統合することで，Github 上のソースコードを Heroku 上で実行中のアプリに容易にデプロイできるようになる．最初に，Heroku 上でアプリケーションを作成し，Heroku ダッシュボードの Deploy タブを押下し，Deploy 方法を選択する箇所から Github を選択する．Github を選択した後は,「Connect to Github」というボタンが表示され，そのボタンを押下する．そうすれ

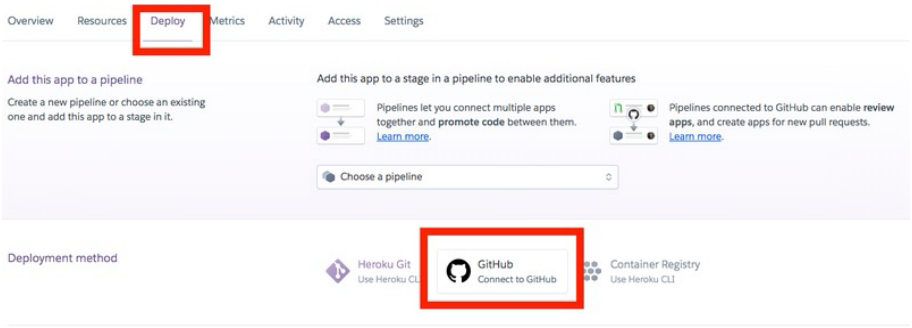

図 3.8: Heroku と Github の連携方法

ば，Github での認証画面が表示される為,「Authorize heroku」というボタンを押下し，認証を成功させる．次に，Github との統合が完了すると図 3.9 のような画面が表示される為，連携したい Github リポジトリを検索し，Connect ボタンを押下する．最後に，任意の

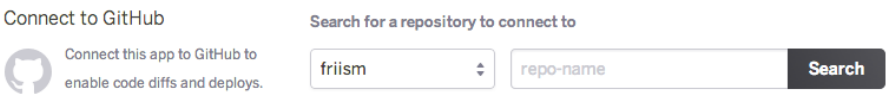

図 3.9: 任意の Github リポジトリと Heroku の連携.

Github リポジトリとの連携が完了すると Manual deploy という欄が表示される．Manual deploy の欄にある「Deploy Branch」というボタンを押下し，Heroku は自動的にビルドを開始する．ビルドに成功した場合はリリースすることができる．

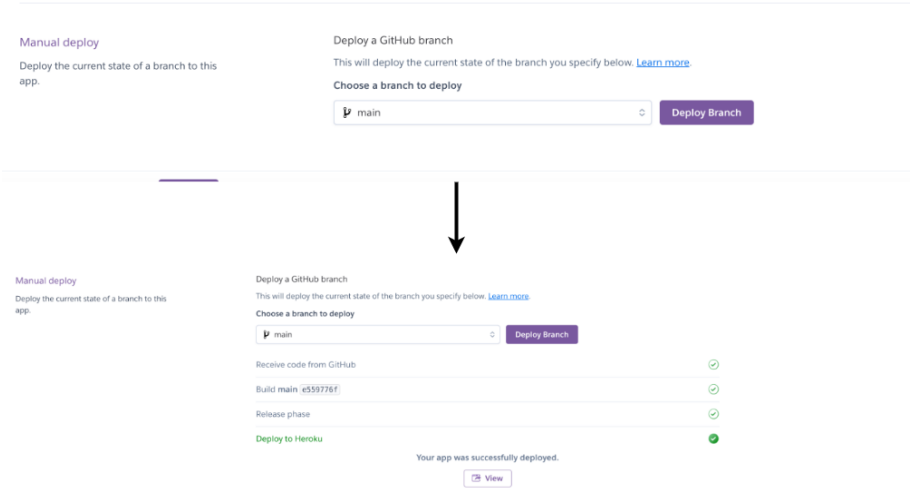

図 3.10: Heroku への手動デプロイ方法

3.3.2 Heroku Add-ons

今回，本番環境用のアドオンとして 2 つのアドオンを選定した．アドオンは Heroku Dashboard または CLI を使用してアプリケーションにインストールすることができるが，Heroku Dashboard を使用してインストールした．1 つ目のアドオンは「Heroku Postgres」である [13]．データベースとのインターフェースを提供する Rails の ActiveRecord[14] のデフォルトが PostgreSQL を対象としている [15]．そこで，Heroku 上でも PostgreSQL を動作させる．次のアドオンは「Heroku Scheduler」である [16]．アーカイブ機能を Heroku 上で実現するために Heroku が提供している専用のスケジューラーアドオンを使用しなければいけない．Rails の場合は規則として rake タスクを設定する必要があり，さらに lib/tasks フォルダ直下にバッチ処理を組み込んだファイルを配置しなければならない．

第4章　まとめ

本研究では，Ruby で書かれたサーバー側 Web アプリケーションフレームワークである Rails を開発環境として，データを共有・見返す作業を癖付けることを目的としたグループウェアアプリ Habit man を開発した．

開発手法として事前に全ての機能やサービスの詳細な要件，作業スケジュールを立てるウォーターフォール開発ではなく，優先順位を付けて重要な機能やサービスを段階に分けて開発し，ユーザーから得た反応から方向性や仮説を見出し，ユーザーへ価値を素早く届け，実戦投入の学びから素早く改善を行うというサイクルを確立するアジャイル開発をした．結果として，研究室のメンバーから早い段階でフィードバックを頂けたのでより良いグループウェアアプリへと修正することができた．また，機能的な側面としてカレンダー表示やタグ付け機能などの複雑な実装はパッケージマネージャーである gem に配布されているライブラリを用いて実現させた．

日付を第 1 キーとしてデータを保存することによって，日付ごとのデータ管理が容易になり，データの検索が高速化した．加えて，データをただ単に溜めていくだけではなく，独自のアルゴリズムから不必要と判定されたデータをアーカイブに移動することによってデータベースやストレージの使用量を最適化できた．

結果として，グループメンバーのデータ共有・見返す作業の習慣化に繋げることができた．

謝辞

本研究を進めるにあたり, 西谷滋人教授には指導教員として終始熱心なご指導を頂きました. 心から感謝いたします. また, アジャイル手法でのアプリ開発を進めていくうえで実際にアプリを使用していただいたり, 貴重な意見を下さった西谷研究室に所属している皆さまにも大変お世話になりました. お礼申し上げます.

参考文献

[1] Discord - https://discord.com (accessd on 21 Nov 2023).

[2] 野口 悠紀雄, 「超」整理法—情報検索と発想の新システム (中公新書), 中央公論新社, (1993).

[3] Slack - https://slack.com/intl/ja-jp/help/articles/115004071768-Slack-%E3%81%A8%E3%81%AF (accessd on 6 Jan 2024).

[4] Agile software development - https://en.wikipedia.org/wiki/Agile_software_development (accessd on 20 Nov 2023).

[5] Ruby on Rails - https://rubyonrails.org/ (accessd on 20 Nov 2023).

[6] まつもとゆきひろ - https://ja.wikipedia.org/wiki/%E3%81%BE%E3%81%A4%E3%82%82%E3%81%A8%E3%82% (accessd on 18 Dec 2023).

[7] Qiita - https://help.qiita.com/ja/articles/qiita (accessd on 18 Dec 2023).

[8] RubyGems - https://rubygems.org/ (accessd on 18 Dec 2023).

[9] Heroku - https://www.heroku.com/what (accessd on 18 Dec 2023).

[10] Heroku Add-ons - https://elements.heroku.com/addons (accessd on 18 Dec 2023).

[11] Convention over Configuration - https://en.wikipedia.org/wiki/Convention_over_configuration (accessd on 18 Dec 2023).

[12] Habit man - https://membermanagementapp-d0c147b97826.herokuapp.com/

[13] Heroku Postgres - https://devcenter.heroku.com/ja/articles/heroku-postgresql (accessd on 29 Jan 2024).

[14] Active Record の基礎 - https://railsguides.jp/active_record_basics.html (accessd on 29 Jan 2024).

[15] Rails コア開発環境の構築方法 - https://railsguides.jp/development_dependencies_install.html (accessd on 29 Jan 2024).

[16] Heroku Scheduler - https://devcenter.heroku.com/ja/articles/scheduler (accessd on 29 Jan 2024).

[17] Strong parameters - https://railsguides.jp/action_controller_overview.html#strong-parameter (accessd on 3 Dec 2023).

付 録A　コード詳細

A.1　データベース(DB)設計

今回，開発したWebアプリのデータベース構造を図A.1に示す．例えばグループには複数のユーザーが所属し，ユーザーは複数のグループに所属できるという仕様にしたためUserテーブルとGroupテーブルの関係性は多対多になる．したがって，中間テーブルとなるGroupUserテーブルを作成した．また，ユーザーは複数のメモを作成することができ，グループは複数のメモを保持するという仕様にしたためGroupテーブルとUserテーブルと同じ関係性のようにGroupテーブルとUserテーブルの中間テーブルとしてMemoテーブルを作成した．

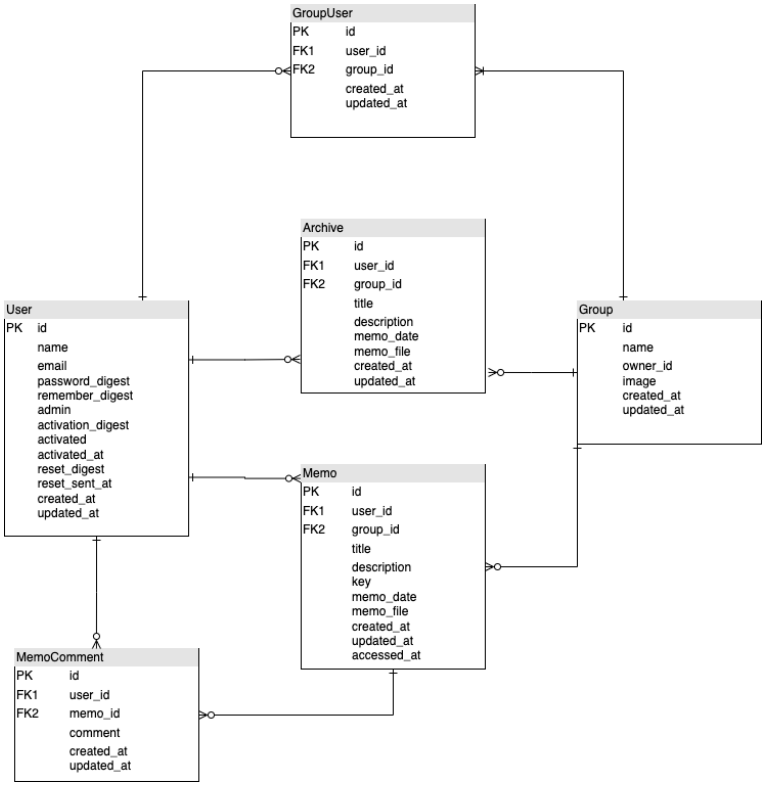

図 A.1: Webアプリの DB設計図.

A.2　グループ関連機能

A.2.1　グループ作成機能

```
class GroupsController < ApplicationController
  def create
    @group = Group.new(group_params)
    @group.owner_id = current_user.id
    @groups = current_user.groups
    if @group.save
      @group.users << current_user
      flash[:success] = "You succeeded in creating new group!"
      redirect_to @group
    else
      render "index", status: :unprocessable_entity
    end
  end

  private
    def group_params
      params.require(:group).permit(:name, :image)
    end
end
```

グループを作成するアクションは create メソッドで行う．group_params メソッドでは strong parameters[17] を明示している．strong parameter とはユーザーがモデルの重要な属性を誤って更新してしまうことを防止するための，より優れたセキュリティ対策のことであり，params メソッドでフォームから送信された情報を受け取り，require メソッドでパラメーターの中にモデルに対応するキーが存在するかを確認し存在する場合にそのバリューを返し，permit メソッドで保存するパラメーターの許可処理を行う．group_params メソッドは Groups コントローラの内部でのみ実行され，Web 経由で外部ユーザーに公開

する必要はないため，Ruby の private キーワードを使って外部から使えないようにする．create アクション内では，@group というインスタンス変数にフォームから送られてきた内容を元に Group モデルのインスタンスが生成されたものを代入する．そして，内容の保存が成功した場合に，グループの中に作成したユーザーを入れ，フラッシュメッセージを表示し，グループの詳細画面へと遷移する．失敗した場合には，エラーメッセージをグループ一覧画面，即ちグループを作成していた画面で表示させる．

A.2.2 メンバー招待機能

```
class GroupsController < ApplicationController
  def invite
    @user = User.find_by(email: params[:group][:email].downcase)
    @group = Group.find(params[:group_id])
    @members = @group.users
    # 招待したメールアドレスがユーザーデータベースに存在すれば招待メールを送信
    if @user
      GroupMailer.invite_member(@group, @user).deliver_now
      flash[:success] = "You succeeded in inviting new member!"
      redirect_back(fallback_location: group_member_path(@group))
    else
      flash.now[:danger] = "Invalid email or Not registered email"
      render "member", status: :unprocessable_entity
    end
  end

  def invite_member(group, user)
    @group = group
    @user = user
    mail(to: @user.email, subject: "Invitation to join the group")
  end
```

```
end
```

メンバーを招待するアクションは invite メソッドで行う．@user というインスタンス変数にはフォームから送られてきたメールアドレスを小文字に変換したものを代入している．そして，そのメールアドレスが登録されているユーザーがデータベースに存在するならば，そのユーザーにメールを通してグループ参加の招待メールを送り，存在しなければ，”Invalid email or Not registered email”というエラーメッセージをメンバー一覧画面に表示させる．

A.2.3　メンバー参加機能

```
class GroupsController < ApplicationController
  def join
    user = User.find(params[:user])
    reset_session
    log_in user
    @group = Group.find(params[:group_id])
    if @group.users.exclude?(user)
      @group.users << user
      flash[:success] = "You succeeded in joing a group!"
      redirect_to group_path(@group)
    else
      redirect_to group_path(@group)
      flash[:warning] = "You have already joined this group"
    end
  end

  def log_in(user)
    session[:user_id] = user.id
    session[:session_token] = user.session_token
```

```
  end
end
```

招待されたメンバーがグループに参加するアクションはjoinメソッドで行う．user変数にはメールのリンクから遷移したユーザー情報を格納している．そして，Rails標準メソッドであるreset_sessionを用いてセッション情報を一度削除してからlog_inメソッドを用いて新たなセッション情報を生成する．条件分岐ではグループ内にメールのリンクから遷移したユーザーが含まれていない場合には新たにグループ内にユーザーを追加するようにし，含まれている場合にはグループの詳細画面に遷移し，”You have already joined this group”というフラッシュメッセージを表示させる．

A.3　メモ関連機能

A.3.1　コメント機能

あるグループメンバーがメモを生成した後に補足情報を追加したい時，あるいは他のグループメンバーがそのメモに対して意見や質問をする際，再度メモを生成するのは利用するグループメンバーの負担が大きくなってしまうリスクがある．そこで，コメント機能を実装することによって，スムーズな情報共有が可能になり，結果としてアプリからユーザー離れが起こりづらくなる．

```
class MemoCommentsController < ApplicationController
  before_action :correct_memo_comment_user, only: [:destroy]


  def create
    @memo = Memo.find(params[:memo_id])
    @group = Group.find(params[:group_id])
    @comment = current_user.memo_comments.new(memo_comment_params)
    @comment.memo_id = @memo.id
    if @comment.save
      # ポストに関わった人たち全員にメールで通知をする
```

```
      NotificationMailer.comment_notification(@memo, @comment, @group).deliver_now
      flash[:success] = "You succeeded in creating new comment!"
      redirect_to group_memo_path(@group, @memo)
    else
      @memo_comment = @comment
      render "memos/show", status: :unprocessable_entity
    end
  end

  def destroy
    MemoComment.find(params[:id]).destroy
    flash[:success] = "comment deleted"
    redirect_to group_memo_path(params[:group_id], params[:memo_id]),
    status: :see_other
  end



  private
    def memo_comment_params
      params.require(:memo_comment).permit(:comment)
    end

    def correct_memo_comment_user
      redirect_to(root_url, status: :see_other) unless
      MemoComment.find(params[:id]).user == current_user
    end
end
```

コメントを作成するアクションはcreateメソッドで行う．memo_comment_paramsメソッドではgroup_params同様にstrong parametersを明示している．createアクション内では，

@comment というインスタンス変数にフォームから送られてきた内容を元に現在ログインしているユーザーが新規コメントを生成したものを代入する. そして，コメントの保存が成功した場合に，メモに関わったグループメンバー全員にコメント内容の通知が送信されるようにしている. 失敗した場合には，エラーメッセージをメモの詳細画面で表示させる.

```
class NotificationMailer < ApplicationMailer
  def comment_notification(memo, comment, group)
    @memo = memo
    @comment = comment
    @group = group
    array_email = []
    @memo.memo_comments.each do |memo_comment|
      array_email.append(memo_comment.user.email)
    end
  array_email.append(@memo.user.email)
    mail bcc: array_email, subject: "#{@comment.comment} from #{@comment.user.name}"
  end
end
```

上記のように，Rails に標準搭載されている Action Mailer を使用すると，アプリケーションのメーラークラスやビューで電子メールを送信できる. 引数として受け取ったメモとコメント，グループをそれぞれインスタンス変数に格納する. そして，メモに紐付けられているコメントユーザー全員のメールアドレスとメモを作成したグループユーザーのメールアドレスを array_email 配列に代入する. 最後に，複数のグループユーザーあてに電子メールを同時送信する際，受取人以外の送信先メールアドレスを伏せて送信することができるブラインドカーボンコピー (bcc) を利用して配列 array_email に格納されたメールアドレス宛に電子メールを送信する.

A.4　検索機能

A.4.1　時間軸検索

時間軸検索とは「超」整理法と呼ばれる新たな整理法の一種である．従来の整理法では，データや書類の内容から分類する「図書館方式」が一般的であった．つまり，整理とは分類であるという考えが古くから定着されている．しかし，従来の整理法にはいくつか問題が生じる．

1. どの分類項目に入れて良いか分からない問題

2. その他問題

まず 1. については，対象となるデータが，複数の内容または属性を持っている場合に，どの分類項目に入れて良いか分からなくなる．例として，pdf の資料をグループ全体に共有しようとした際に，pdf という項目に入れるのか，その pdf 資料の内容に関する項目に入れるのかという問題が生じてしまう．個人的にデータを管理するのならば，図書館方式でも良いかもしれないが，グループとしてデータを管理していくのならば，グループ 1 人 1 人が共通の分類法を身につけなければならない．これは非常に危険であり，非効率である．次に 2. についてはデータはどの分類項目に入らないものもある．この場合には,「その他」などといった分類項目に残しておくというのが，ごく常識的な対処であろう．しかし，これこそが最大の陥穽なのであり,「その他」はハードルが低く便利な分類項目だから，どんどんデータが入ってくる．その結果，とどまるところをしらず膨れ上がり，収拾がつかなくなる．また，データを分類した瞬間は正しい項目に入れておいたか覚えていたとしても，時間が経過すればどの項目に入れてしまったか忘れてしまうこともある．使用頻度が低いデータは特にそうである．そこで時間軸による検索は，極めて有効的である．理由として以下の 2 点が挙げられる．

1. 使用する書類，データの大部分は，最近使ったものの再使用である．

2. 人間の記憶は，時間順に関しては強い．

A.4.2　タグ検索

タグ検索は日付キーを忘れて時間軸検索ができない場合に用いることを想定する．図のように共有されたデータ一覧画面の検索欄からタグとなる単語を入力し該当するタグに紐付けられたデータを絞り込むことができる．また，1つ1つの投稿データのタグリンクをクリックすることによっても検索することができる．さらに動画のリンクや音声・画像ファイル共有した場合にはそれらの内容の特徴を表したタグを付けておくことによってデータの検索を容易にすることもできる．

図 A.2: タグでデータを絞り込む一連の動作．

A.4.3　キーワード検索

キーワード検索はタグ付けがされていないメモ，タグを忘れてしまったメモを検索する場合に用いることを想定する．タグ検索と同様データ一覧画面の検索欄からメモのタイト

ルを入力し単語がヒットすれば該当するデータを絞り込むことができる.

```
class GroupsController < ApplicationController
  def show
    @group = Group.find(params[:id])
    @memo = @group.memos.new
    @pagy, @memos = pagy(@group.memos.order(updated_at: :desc).limit(20))
    if params[:key_word]
      @pagy, @memos = pagy(@memos.where("title LIKE ?", "%#{params[:key_word]}%"))
      unless @memos.present?
        @pagy, @memos = pagy(@group.memos.tagged_with(params[:key_word]))
      end
    end
  end
end
```

key_word というパラメータを受け取った時にのみ，Rails の検索用メソッドである where と like 句を用いて受け取ったパラメータからタイトルにヒットするデータを絞り込み@memos というインスタンス変数に格納する． その後，@memos の中身が存在するかどうかを present メソッドで判定し，存在しなければ受け取ったパラメータからタグ検索をする.

A.5 アーカイブ機能

lib/tasks/create_archive.rake

```
namespace :create_archive do
  desc "アーカイブの定期実行プログラム"
  task add_archive: :environment do
    Group.all.each do |group|
      group.memos.each do |memo|
        if memo.accessed_at == nil ||
        (memo.accessed_at < 2.weeks.ago && memo.key == false)
```

```
          archive = group.archives.new(
                                        title: memo.title,
                                        description: memo.description,
                                        user_id: memo.user_id,
                                        group_id: memo.group_id,
                                        memo_date: memo.memo_date,
                                        memo_file: memo.memo_file
                                       )
          archive.tag_list << memo.tag_list
          archive.save
          memo.destroy
        end
      end
    end
  end
end
```

グループのメモ1件1件に対してアクセスされた日時を表すaccessed_atがnilまたは2週間以上経過しており，keyカラムがfalseの場合に新たにArchiveインスタンスを生成し，保存する．そして，アーカイブに移動することになったメモは削除される．これにより，アクティブなデータとアーカイブデータを分離することが可能になる．これを毎日0時に定期実行されるようにバッチ処理として組み込む．

A.6 通知機能

```
class NotificationMailer < ApplicationMailer
  def notification_for_member(notification, group)
    @notification = notification
    @group = group
    mail bcc: @group.users.pluck(:email),
```

```
    subject: "#{@notification.title} from #{@notification.user.name}"
  end
end
```

上記のように，Rails に標準搭載されている Action Mailer を使用すると，アプリケーションのメーラークラスやビューで電子メールを送信できる．引数として受け取ったメモとグループをそれぞれインスタンス変数に格納する．そして，複数のグループユーザーあてに電子メールを同時送信する際，受取人以外の送信先メールアドレスを伏せて送信することができるブラインドカーボンコピー (bcc) を利用して電子メールを送信する．