筑波大学大学院博士課程

システム情報工学研究科特定課題研究報告書

写真共有サービスの
ユーザ動向可視化・分析システムの開発
―サーバサイドの設計・実装―

古谷 翔太

修士（工学）

（コンピュータサイエンス専攻）

指導教員　田中 二郎

2016年3月

概要

RoomClip はインテリアの写真共有に特化したソーシャル・ネットワーキング・サービス（SNS）である。RoomClip を開発・運営している Tunnel 株式会社では、近年さらに活発なユーザを増やそうとさまざまな施策を講じている。その上で、ユーザに対して行う施策に関する知見を得ることおよび施策の効果を確認することに対して、有効な分析手段を準備したいと考えている。そこで、この研究開発プロジェクトでは、RoomClip の利用データを用いて、ユーザの動向を分析できるツールを開発した。筆者は、指定した期間で RoomClip に登録したユーザの行動がどのように変化したかを確認できる画面の開発、本ツールで利用するユーザデータを取得するための API の実装およびプロジェクト全体のマネジメントを行った。本報告書では、プロジェクトの概要、既存の分析ツールの調査結果、システム全体の構成および筆者が担当した開発部分やマネジメントの詳細について述べる。

目 次

図 目 次

表 目 次

第1章　はじめに

本報告書は、筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻の高度IT人材育成のための実践的ソフトウェア開発専修プログラム（以後、高度ITコース）における研究開発プロジェクトとして実施した「写真共有サービスのユーザ動向可視化・分析システムの開発」の成果を述べたものである。

ソーシャル・ネットワーキング・サービス（SNS）は、人と人との社会的ネットワークをインターネット上で構築するためのサービスであり、趣味、地域、文化など、多種多様な分野のSNSが存在している。総務省の平成26年度版情報通信白書[1]によると、近年、10代から20代の若年層を中心に、SNSの総数および利用者数が増加傾向にある。特にLINE[*1]やFacebook[*2]などの著名なSNSでは、20代の利用率がLINEで80.3％、Facebookで57.0％など、非常に高い利用率を示しており、SNSは現代の社会生活を送る上で欠かせないものになりつつあると考えられる。

衣食住の分野においても、SNSが数多く存在している。衣食住は人の生活の基礎にあたる部分であり、多くの人が関心を持ち、消費活動を行っている分野である。これらのSNSでは、他のユーザのスタイルを参考にし自分のスタイルに取り入れるなどの行動も見られる。鳥海ら[2]や松尾ら[3]によると、実際の友人間のネットワークに限定しない包括的なSNSの場合、中心的なユーザが積極的に関与することによりSNS全体の活性化がなされているため、中心的なユーザが衣食住の特定の企業の商品を使ったスタイルを示すことにより、他のユーザのそれらの商品に対する消費活動を促すという、衣食住の分野のマーケットに対する好影響も考えられる。高木[4]によると、商品に対するユーザレビューとそれの賛同コメントは購買に大きな影響を与える。そのため、SNSにおいても、インテリアに対するユーザによるレビューは、商品販売を促進すると考えられる。

Tunnel株式会社[*3]が2012年初頭から運営するRoomClip[*4]は、衣食住の中の「住」にあたる、インテリアに関するSNSの1つであり、インテリアに関する写真を投稿し、他のユーザとの交流や、コンテストへの出場などができる写真共有サービスである。現在、Webブラウザまたは AndroidとiOS向けに提供されている専用アプリケーションから利用することができる。

RoomClipは日本のインテリアに関するSNSの中では最大規模であり、登録ユーザ数は30万人以上、月間PV数は9500万PV以上、投稿された写真数は100万枚を超える。登録ユー

[*1] http://line.me/
[*2] https://www.facebook.com/
[*3] http://www.tunn-el.com/
[*4] http://roomclip.jp/

ザ数や投稿写真数は増加傾向にあり、インテリアに関心のあるユーザ層を中心に、今後さらに規模が大きくなることが予想される。また、インテリア販売企業とのタイアップも頻繁に行われており、前述の販売促進による広告効果も期待されている。

SNS を運営する上で最も重要なことは、新規ユーザを獲得することと、既存のユーザを維持することである。Tunnel 株式会社では、特に既存のユーザの維持に関し、Android と iOS のアプリケーションを利用しているユーザにはコンテストに関するプッシュ通知を定期的に送るなど、ユーザがより積極的に RoomClip を利用するよう施策を講じている。

しかし、現在 Tunnel 株式会社では、RoomClip ユーザの利用データに対する継続的な分析が行われていない。理由としては、RoomClip のユーザの動向を包括的に分析できるツールが存在しないことと、継続的にデータベースを直接参照し分析を行うコストが高いことが挙げられる。したがって、現状では、既存のユーザの維持に関して、どのユーザに対して重点的に施策を講じるべきかがわからず、施策の効果も確認することが難しい。

そこで筆者が携わった研究開発プロジェクト（以後、本プロジェクト）では、Tunnel 株式会社が RoomClip の利用データからユーザの動向を把握することができるツールを開発することにした。ツールで用いる分析手法および分析結果の見せ方は、本プロジェクトに参加するメンバ内で議論し、Tunnel 株式会社に提案を行った。提案内容は Tunnel 株式会社を交えて検討・取捨選択し実装した。

筆者は、本プロジェクトに開発メンバとして参画し、指定した期間で RoomClip に登録したユーザの行動がどのように変化したかを確認できる画面の開発、アプリケーションサーバの構築、および RoomClip の利用データを提供する WebAPI の実装を行った。また、開発メンバのリーダとして、本プロジェクト全体のスケジュールマネジメントおよびタスクマネジメントも平行して行った。

本報告書は、本文全 8 章と、開発ドキュメントなどの付録で構成されている。第 2 章では、プロジェクトの概要として、顧客の問題点、本プロジェクトで開発するツールへの要求およびプロジェクトの体制について述べる。第 3 章では、本プロジェクトで SNS 分析ツールを開発するにあたって、既存の SNS 分析ツールの調査結果を述べる。第 4 章では、本プロジェクトで開発したツールの概要について述べる。第 5 章と第 6 章では、筆者が担当した開発箇所の内容、詳細および考察について述べる。第 7 章では、筆者がプロジェクトリーダとして本プロジェクトのマネジメントを行った詳細について述べる。第 8 章では、本プロジェクトの振り返りと今後の展望を述べ、まとめとする。

第2章　プロジェクトの概要

本章では、本プロジェクト開発背景およびプロジェクトの体制について述べる。

2.1　顧客の問題点と要求

本プロジェクトは、東京都文京区に所在する Tunnel 株式会社を顧客およびエンドユーザとする。Tunnel 株式会社は 2012 年 5 月に RoomClip をリリースし、今日に至るまで規模を拡大させ続け、現在もユーザ数や写真投稿数が増加傾向にある。しかし、今後もより RoomClip を発展させるためにさまざまな施策を講じているが、以下のような問題が発生している。

施策の効果が確認できない
: ある一定期間で既存の全ユーザを対象に、より活発に RoomClip を利用するためのキャンペーンなどを行った場合、既存のユーザがより活発に利用し始めたり、登録したものの長期間利用していなかったユーザが再び使い始めたりしたのかどうかを現状では把握することができない。

どのユーザに施策を講じるべきかわからない
: できる限り RoomClip を使わなくなるユーザを減らすため、使わなくなりそうなユーザに対しあらかじめ対策を講じたいが、どのユーザが使わなくなりそうか現状では大まかにしか把握することができず、どのユーザに対して対策を講じれば効果的かを知ることができない。

現在は、データベースを直接参照したり、Google Analytics[*1] などのアクセス解析ツールを参照したりすることで分析を行い、問題の解決を図ろうとしているが、知見に結びつく兆候を発見することが難しく、分析するコストが高いため、継続的な分析に苦慮し、問題の解決がなされていない。

このような問題から、Tunnel 株式会社は、RoomClip の利用データを継続的に分析することができ、ユーザに関する何らかの知見を得ることができるツール、つまり、RoomClip の利用データから、これから使わなくなりそうなユーザの兆候や、これから中心的な役割を担いそうなユーザの兆候など、積極的に使うユーザとそうでないユーザのパターンを見出すことができるツールを要求している。ツール自体に関しては、なるべく簡単かつ効果的に分析を行うことができるようなインタフェースを、プラットフォームへの依存が少ない Web アプリケーション上で実装することが要求である。

[*1] https://www.google.com/analytics/

2.2 目的

本プロジェクトは、2.1 節で挙げた問題点を解決し、顧客の要求を満たすため、RoomClip の利用データをユーザの観点から分析できるツールを開発することが目的である。

ツール内で利用する分析手法および分析結果の見せ方は、プロジェクトのチーム内で議論し、Tunnel 株式会社に提案を行った。提案内容は、チーム内と課題担当教員の間で議論をした上で、Tunnel 株式会社を交えて検討し、取捨選択しツール内に実装した。

なお、本プロジェクトで開発するツール名は、Analytics for RoomClip から ARC と名付けた。以後、本プロジェクトで開発するツールは ARC と呼称する。

2.3 体制とステークホルダ

本プロジェクトは、高度 IT コースに所属する筆者および河越嵩介、崔野、万シャンシャンの 4 名を開発メンバーとしたチームで開発を行った。開発にあたり、1 週間に 1 回程度ミーティングを開催し、課題担当教員の岡瑞起准教授および橋本康弘助教に対し進捗報告を行い、指導・助言を受けた。なお、開発チーム内の役割分担は、プロジェクト当初から決定はせず、開発の方針が決定した時点で分担を行った。詳細は第 7 章で述べる。

ARC の分析対象となるデータとして、本プロジェクトの顧客かつ ARC のエンドユーザである Tunnel 株式会社から、RoomClip の利用データの提供を受けた。また Tunnel 株式会社には、実際に 7 月 9 日、10 月 6 日、11 月 27 日にそれぞれ訪問し、代表取締役の高重正彦氏、執行役員の平山知宏氏に対し ARC の開発状況の報告を行い、フィードバックを受けた。

図 2.1 に本プロジェクトの体制を示す。

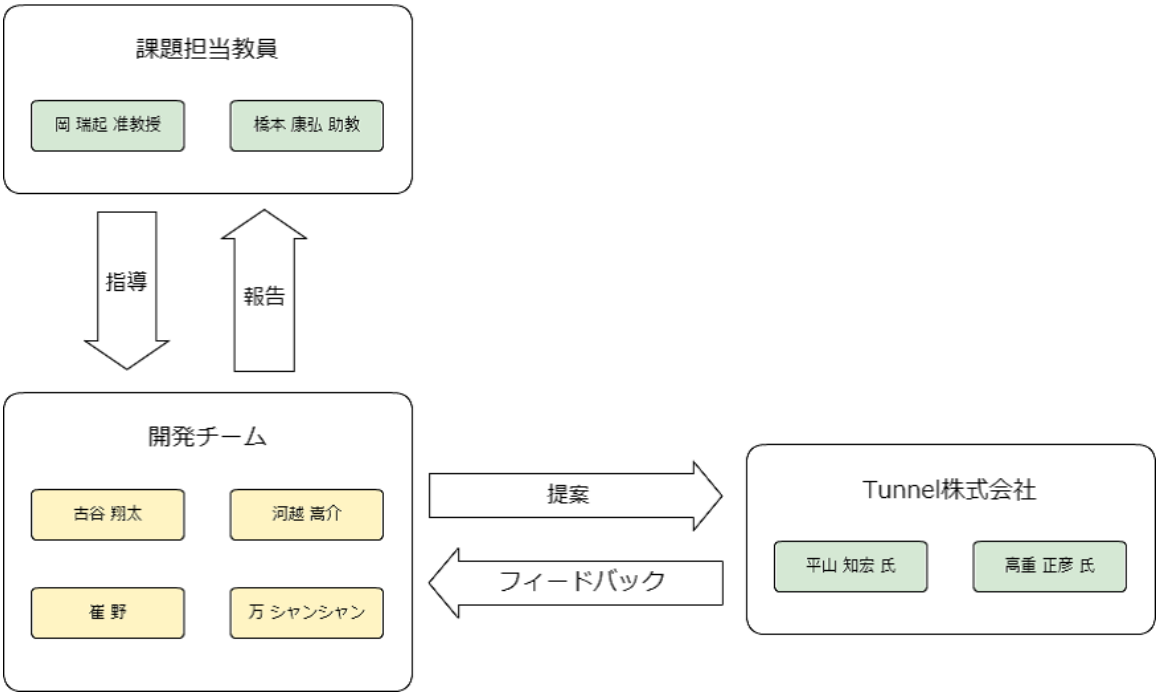

課題担当教員
岡 瑞起 准教授
橋本 康弘 助教
指導
報告
開発チーム
古谷 翔太
河越 高介
崔 野
万 シャンシャン
提案
フィードバック
Tunnel株式会社
平山 知宏 氏
高重 正彦 氏

図 2.1: ステークホルダ

第3章　既存の分析ツールの調査

ARC を開発するにあたって、データの可視化手法やユーザインタフェース（UI）を参考にし、ARC へのイメージを固めるため、SNS を分析することができる既存のデータ分析ツールの調査を行った。調査方法としては、チームメンバーがそれぞれ調査を行い、ミーティング時に各自で調査結果を報告し、議論しまとめるという方法をとった。

その結果、既存の分析ツールは、以下の 2 種類に大別できることがわかった。

1. 汎用データ分析ツール
   さまざまなデータの種類や形式に対応したツールであり、データの見せ方も豊富で、ユーザが自由に指定することができる。

2. 特化型分析ツール
   特定の SNS に特化した分析を行うツールであり、分析できるデータ形式や可視化手法もある程度固定されている。

3.1 節と 3.2 節で、それぞれの特徴や具体例について述べる。

3.1　汎用データ分析ツール

汎用データ分析ツールは、いわゆるセルフサービスビジネスインテリジェンスツール（セルフサービス BI ツール）のことであり、主にビジネス上何らかの知見を得るために、さまざまな業界や業種で利用されている。

ほとんどの汎用データ分析ツールは、CSV などの形式で一連のデータをインポートし、それをグラフなどの可視化手法を用いて描画する。Microsoft Excel などの表計算ソフトウェアにデータを記述し、描画機能を用いてグラフを描画するよりも、基本的なクリックだけでグラフの描画をすることができ、描画の調節なども直感的に行うことができる。

また、汎用データ分析ツールは、データ処理に関する特別な知識がなくても利用することができるため、企業内では、情報システム部門などのデータを専門に扱う部門だけではなく、経営層が意思決定のために用いることもある [5]。しかし、一般的な汎用データ分析ツールは、ツール自体が分析を行うのではなく、あくまでデータに関するレポーティングに特化したツールであり、ツール自体が何らかの知見を算出するものではない。また、分析するデータは、ツールが指定した形式でユーザが準備する必要がある。SNS で利用する場合は、ユーザ数や投稿数などのデータをデータベースから用意し、それをツールに入力するという流れになる。

著名な汎用データ分析ツールの例としては、Tableau Desktop[*1]、TIBCO Spotfire[*2]、Qlik Sense[*3] などが挙げられる。また、分析だけではなく予測を行うものとして、IBM Watson Analytics [*4] なども提供され始めている。

3.2 特化型分析ツール

特化型分析ツールは、Twitter[*5] や Instagram[*6] など、特定の SNS に対してのみ分析を行うツールであり、個人が自分の SNS アカウントに関して分析を行ったり、企業が広報用などの SNS アカウントに対して分析を行ったりするために用いるのが大きな特徴である。

汎用データ分析ツールとの共通点は、データ分析に関する専門知識がなくても分析が行えることである。特化型分析ツールの場合は、さらに、ほとんどの特化型分析ツールは特定の SNS のアカウントと連携するだけで分析を行うため、ユーザが自らデータをやりとりする必要はないため、データ加工などの手間が省けるという利点がある。

特化型分析ツールでは、分析できるデータは、自分の投稿に対する PV 数や自分の投稿をお気に入りに登録した数など、ある程度分析ツール側で指定されていて、データに対する自由度は無い場合が多い。また、データの可視化手法も、ユーザがデータに合わせて選択するのではなく、PV 数ならば折れ線グラフなど、そのデータに最適な可視化手法があらかじめ指定されていることが多い。

著名な特化型分析ツールの例としては、Twitter に対する Twitter Analytics[*7]、Instagram に対する ICONOSQUARE[*8] などが挙げられる。Google Analytics などの Web 解析ツールは、特定の SNS を対象にはしていないものの、Web サイトに対して、データ加工をせずに分析を行えるという点で、特化型分析ツールに属すると考えられる。

3.3 調査結果を踏まえた開発の方針

既存の分析ツールを調査した結果、汎用データツールと特化型分析ツールの 2 種類があることがわかった。ARC は、これら 2 種類の長所を参考にし、以下のような特徴を備えるツールを目指すことにした。

- ユーザはデータを用意する必要がない
  ARC 専用データベースを構築し、加工した利用データを保存することにより、ユーザが RoomClip のデータベースから利用データを用意し加工する必要をなくす。

[*1] http://www.tableau.com/products/desktop
[*2] http://spotfire.tibco.com/
[*3] http://www.qlik.com/products/qlik-sense
[*4] http://www.ibm.com/software/jp/cmp/watsonanalytics/
[*5] https://twitter.com/
[*6] https://www.instagram.com/
[*7] https://analytics.twitter.com/
[*8] http://iconosquare.com/

- UI が平易でわかりやすい
  複雑なインタフェースをなるべく避け、他の分析ツールと同じような使用感とする。

- あらかじめ可視化手法が用意されている
  あらかじめ ARC を使用するシーンを想定し、それに適した可視化手法を採用する。採用した可視化手法に関しては、顧客である Tunnel 株式会社と適宜報告しフィードバックをもらい、チームからの一方的な提案にならないよう、常に意思疎通を図る。

このような特徴を持つツールとして ARC を開発することにした。第 4 章から具体的な開発内容について述べる。

第4章　開発したツールの概要

本章では、ARC を開発するにあたってのアプローチと、チーム内で議論を重ねた上で設計・実装した ARC の全体像について述べる。

4.1　開発のアプローチ

ARC では、ユーザの動向を分析するため、現在 30 万人以上いるユーザをいくつかのグループに分類し、それらの分類に関して、経時変化を定量的・定性的に可視化することにした。本節では、その分類の方法と ARC で用いるノードという概念について述べる。

4.1.1　ユーザの分類

ユーザが RoomClip 上で起こせる行動の中で、特に投稿された写真に対する行動のことをアクション、1 回以上のアクションの集まりをユーザアクティビティと呼ぶことにする。Tunnel 株式会社としては、ユーザ全体のユーザアクティビティを活発にすることが最優先課題であり、一人でも多くのユーザにアクションを起こしてほしいという要望がある。

現在、アクションは以下の 7 種類に絞ることができる。

1. 他のユーザが投稿した写真を閲覧する
2. 他のユーザが投稿した写真の「いいね！」ボタンを押す
3. 他のユーザが投稿した写真をお気に入りに追加する
4. 他のユーザをフォローする
5. 他のユーザが投稿した写真に対してコメントを付ける
6. 写真を投稿する
7. タグをお気に入りに追加する（現時点では Android・iOS 向けアプリケーション版のみ対応）

この 1. から 7. に対して、それぞれアクション名として名前を定義する。アクション名を図 4.1 に示す。

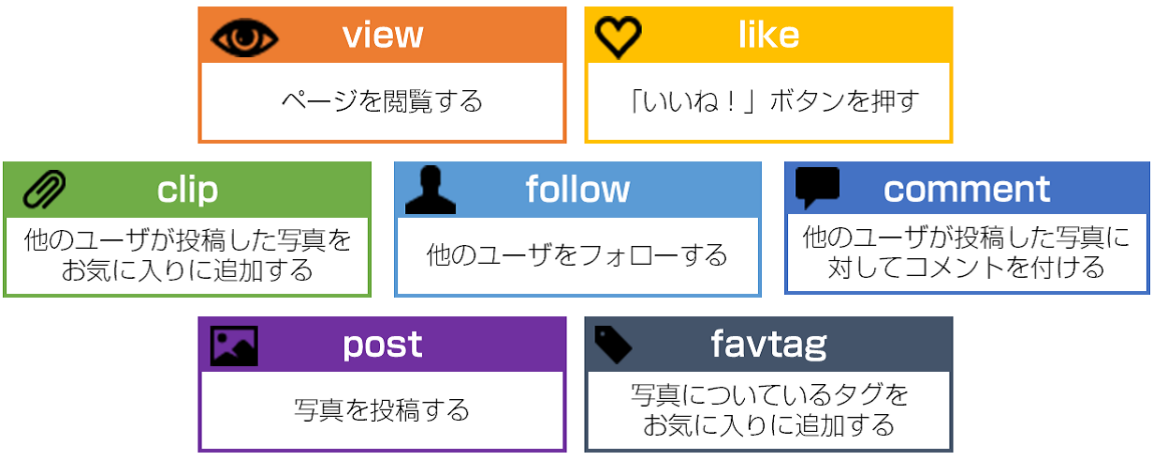


図 4.1: 7 つのアクション

現在、ユーザアクティビティを観点として、ユーザの行動パターンからどのようなユーザが RoomClip を使わなくなるかの分析を行っているが、未だ明らかにされていない。また、他ユーザに影響を与える中心的なユーザ、ページを見ているだけで写真に対するアクションは行わないいわゆる「ROM 専」、まったくページを見ない非アクティブなユーザなどユーザのおおまかな分類はなされているが、現在ははっきりとした定義がなされていない。そこで、ARC では、アクションを観点として、ユーザを分類することにする。4.1.2 節で、ユーザの分類について述べる。

4.1.2 ノードについて

4.1.1 節で述べた 7 つのアクションを用いて、一定期間にそのアクションを行ったか行っていないかの二値で分類することを考える。7 つのアクションのうち VIEW を行っていないユーザは、他のアクションも行っていないと仮定できる。したがって VIEW を行っていて LIKE・CLIP・FOLLOW・COMMENT・POST・FAVTAG の 6 種類を行ったか行っていないかの分類と、VIEW をしていない分類の 2 グループに分けることができる。分類は、VIEW を行っているグループで $2^6 = 64$ 通り、VIEW を行っていないグループが 1 通りなので、合計 65 通り存在する。ARC では、これらの分類をノードと呼ぶことにする。このノードに関して、ノード間または時系列のノードの変動を分析するようなツールとして ARC を開発する。

例として、1 月 1 日から 1 月 7 日まで VIEW を 20 回、COMMENT を 10 回、POST を 1 回行い、他のアクションは起こしていないユーザを考える。これらを各アクションに関して行ったか行っていないかで二値化すると VIEW、COMMENT、POST がアクションあり、その他がアクションなしとなる。したがってこのユーザは、図 4.2 で示すように、1 月 1 日から 1 月 7 日において、VIEW あり、LIKE なし、CLIP なし、FOLLOW なし、COMMENT あり、POST

なし、FAVTAG なし、というノードに属することになる。このような分類をすべてのユーザに対して行い、すべてのユーザはいずれかのノードに属するようになる。

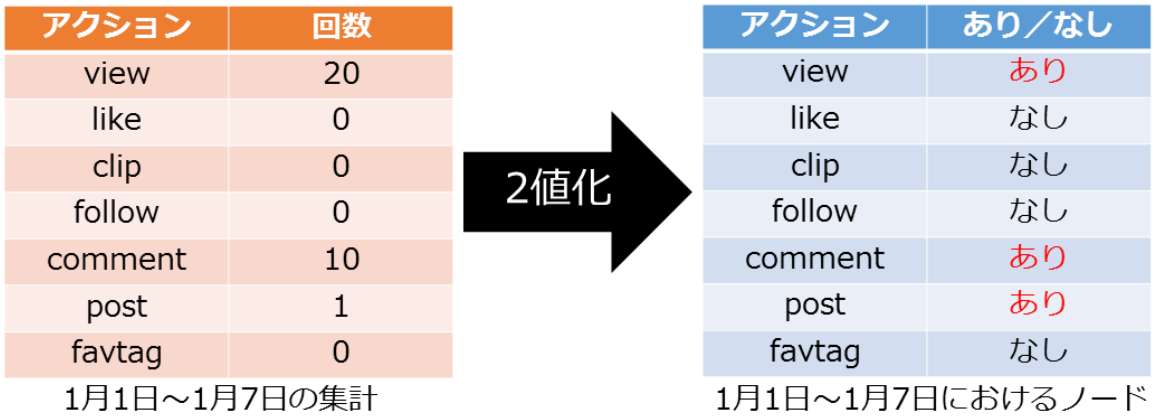


図 4.2: ノード分けの例

4.2 分析対象のデータ

ARC の分析対象となるデータは、Amazon Web Service（AWS）[*1] 上で運用されているデータベースに含まれている RoomClip の利用データである。この利用データのうち、Tunnel 株式会社から提供された利用データは、データベースのダンプファイルの一部と、ログデータの一部である。

ダンプファイルの構成は表 4.1 のようになっている。さらに、ユーザの詳細な情報として、表 4.2 に示すテーブルも用意されている。このダンプファイルはサービス開始日である 2012 年 4 月から 2015 年 5 月 21 日までのデータが含まれている。

ログデータは以下のような構成となっている。

1. 写真に対するビューのログデータ（photo テーブル）
   集計期間 : 2014-10-07 から 2015-08-01
   写真の詳細画面にアクセスしたログデータ。アクセス日時、アクセスしたユーザ ID およびアクセスされた写真 ID の情報がリアルタイムで記録される。

2. ページビューのログデータ（access テーブル）
   集計期間 : 2014-10-07 から 2015-08-01
   Web のトップ、写真の詳細画面、タイムラインにアクセスしたログデータ。アクセス日時とどのページにアクセスしたかの情報がリアルタイムで記録される。

[*1] https://aws.amazon.com/

表 4.1: RoomClip ダンプファイルの構成

テーブル名	テーブルの説明	カラム名	カラムの説明
user_info	ユーザの基本情報	created	アカウント作成日時
		id	ユーザ ID
		name	名前
		gender	性別
		birth	誕生年
user_follow	ユーザのフォロー	created	フォロー日時
		following_id	フォローしたユーザ ID
		followed_id	フォローされたユーザ ID
tag	タグの付加	created	登録日時
		id	タグ ID
		name	タグ名
photos	写真の投稿	created	写真投稿日時
		id	写真 ID
		taker_id	投稿者のユーザ ID
photo_clips	写真のクリップ	created	クリップ日時
		user_id	クリップしたユーザ ID
		photo_id	クリップした写真 ID
photo_comment	写真に対するコメント	created	コメント日時
		photo_id	コメントがついた写真 ID
		user_id	コメントしたユーザ ID
		comment	コメント本文
tag_photo_relation	タグと写真の関連	created	タグ付加日時
		photo_id	写真 ID
		tag_id	付加されたタグ ID
RoomClip_photo _tag_timeseries	時間ごとにまとめた 写真とタグの関連	time	投稿日時
		photo_id	写真 ID
		taker_id	投稿者のユーザ ID
		tag_ids	付加されたタグ ID の配列

表 4.2: ユーザの詳細情報テーブル

テーブル名	テーブルの説明	カラム名	カラムの説明
user_info_detail	ユーザの詳細情報	created	アカウント作成日時
		id	ユーザ ID
		name	ユーザ名
		room_number	部屋 ID（通常ユーザ ID と同じ）
		country	国籍
		region	居住地域
		style	一人暮らしなどの居住スタイル
		job	職業
		gender	性別
		birth	誕生年
		layout	3LDK などの部屋のレイアウト
		area	部屋の面積

4.3 データベースの構成

ARC では、表 4.1 と 4.2 に示したデータとログデータを、4.1 節で述べたノード分類がしやすいように加工し、専用のデータベースに保存する。なお、ユーザデータは表 4.1 で述べた user_info テーブルをそのまま利用することとした。

データベースは RoomClip と同じく AWS の Redshift を利用する。Redshift は PostgreSQL をベースとしたデータウェアハウスサービスであり、接続は PostgreSQL 経由で行うことができる。Redshift の特徴は、列指向データベースとして設計されており、特定の列を取り出して集計するという操作が得意ということが挙げられる。したがって、ユーザの利用データのような、列に対して集計を行うことが多いデータと相性が良い。しかし、Redshift は従量課金制であるため、開発期間中は、1 年間無料で利用することができる RDS で代用した。

加工したデータは user_activity という 1 つのテーブルに格納することとした。このテーブルでは、どのユーザがどのアクションを何回行ったかを 1 日単位で保存する。表 4.3 に user_activity カラム名とそれの説明を示す。

また、各ノードにはノード ID という一意に対応する ID を付与した。したがって、ノード ID とノードの対応を取るために、node テーブルも作成した。node テーブルでは、ノード ID と 7 つの各アクションの二値判定を行うカラムを持ち、各行では、各アクションをしたかしていないかを、0 か 1 かの二値で表し、それに対するノード ID が示されている。例えば、ノード ID が 65 の行は、VIEW から FAVTAG までのすべてのアクションの欄が 0 となっている。これらの対応は、ノード ID 対応表としてドキュメント化した。本報告書の付録として対応表を添付する。表 4.4 に node テーブルのカラム名と説明を示す。

表 4.3: user_activity テーブル

カラム名	説明
u_date	日付（YYYY-MM-DD 形式）
u_id	ユーザ ID
u_view	VIEW の数
u_like	LIKE の数
u_clip	CLIP の数
u_follow	FOLLOW の数
u_comment	COMMENT の数
u_post	POST の数
u_favtag	FAVTAG の数

表 4.4: node テーブル

カラム名	説明
n_id	ノード ID
n_view	VIEW の有無
n_like	LIKE の有無
n_clip	CLIP の有無
n_follow	FOLLOW の有無
n_comment	COMMENT の有無
n_post	POST の有無
n_favtag	FAVTAG の有無

4.4　システム構成図

ARC 全体のシステム構成図は、図 4.3 のようになる。ARC のシステムはマイクロサービスアーキテクチャ[6] のエッセンスを取り入れ、データを整形し可視化する部分とデータを提供する部分をフロントエンドとバックエンドとして分離し、フロントエンド担当者とバックエンド担当者を独立させて開発を行うことができるようにした。また、フロントエンドとバックエンドは WebAPI のみによって連携を行うようにするため、フロントエンドとバックエンドの内部の動作はお互いに影響しない。サーバはフロントエンドとバックエンド共に、AWS の EC2 のインスタンスを利用している。

クライアントはブラウザの Google Chrome を利用して、ARC のフロントエンド部分にアクセスする。フロントエンドではあらかじめ用意した HTML、JavaScript、CSS ファイルを nginx が HTTP サーバとして提供する。

バックエンドサーバでは、データベースからデータを取得し、それを WebAPI として提供する。API は、データ分析ライブラリの豊富さ、フレームワークとしての手軽さから、Python の Web フレームワークの一つである Pyramid[*2] を選択し利用した。またバックエンドサーバ上では、将来的にアクティブユーザの予測機能を提供する WebAPI も用意し、本プロジェクトも、用意されたデータを用いた機械学習による予測結果を提供することにした。

データベースは AWS の Redshift（開発中は RDS）上で動作している PostgreSQL を利用しており、バックエンドサーバから Python のライブラリを通じてアクセス可能としている。データベースに保存するデータは、同じく AWS 上の S3 に保存された RoomClip の未加工の利用データを手動で整形して保存している。

4.5　可視化手法と各画面

ARC で提供する RoomClip 利用データの可視化に関して、Tunnel 株式会社の要望を踏まえ、チーム内や課題担当教員と議論を重ねた上で、以下の 3 つの画面が必要と判断した。

1. 全体俯瞰図
   期間を指定し、全ユーザの動向を知ることができる画面

2. 個別比較図
   指定した 2 つの期間でユーザ数がどのように変化したかを知ることができる画面

3. トップページ
   ユーザ数やアクション数の変化をひと目で大まかに知ることができる画面

ARC では、全体俯瞰図を Flow 画面、個別比較図を Ranking 画面、トップページの変化を大まかに知る部分は Dashboard 画面と名付け開発を行った。各画面共に、紙や Microsoft Excel 上などで作成したプロトタイプを共有し、相互フィードバックを行う形で作成した。これら

[*2] http://www.pylonsproject.org/

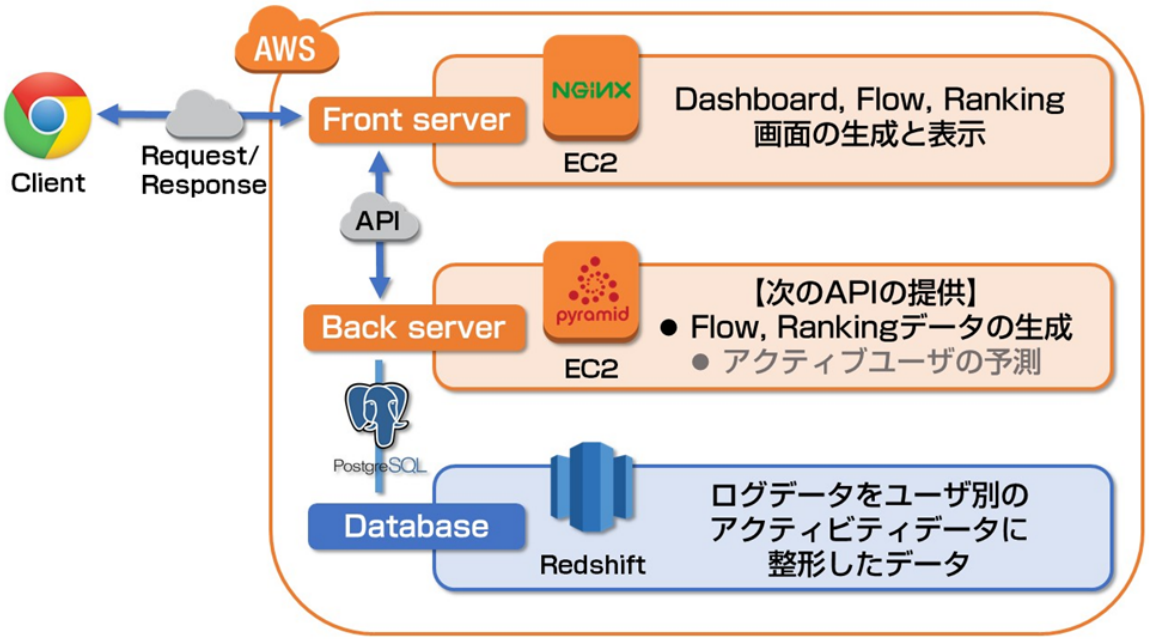


図 4.3: システム概要図

の画面は、ARC のフロントエンドにブラウザでアクセスすることによって表示することができる。本節では、全体の概要として、各画面の意図および最終的に出来上がった実際の画面のスクリーンショットを示す。

4.5.1 トップページ

ARC のトップページにあたる画面を 4.4 に示す。ここでは昨日の新規登録者と現在の総ユーザ数を見ることができると同時に、3 つの画面にジャンプすることができる。

4.5.2 Flow 画面

図 4.5 で示す Flow 画面は、ノード間の流出入を表す画面である。工程間の流量を表すサンキーダイアグラムを利用し、ある期間とある期間の間で、ノード間でどの程度の人数が移動したのかを視覚的に表現している。流出入量を見る期間のことを、ここではレイヤと呼ぶ。レイヤ数は、現在 2 から 4 の間で設定することができる。したがって最大で 4 つの期間でそれぞれ流出入量を見ることができる。この画面では、65 種類のノードに加えて、各レイヤで RoomClip に登録した、新規登録ユーザのノードも追加されている。レイヤは統計開始日から、統計範囲に示されている日数で範囲が決められる。例えば統計開始日を 2015 年 1 月 1 日、統計範囲を 7 日、レイヤ数を 3 に設定した場合、以下のようにレイヤが設定される。


RoomClip ARC
Dashboard
Ranking
Flow
ARC
Analytics for RoomClip
昨日の新規登録者人
総ユーザ数人
Ranking
Flow
Dashboard
二つの期間の変動率
ユーザ行動の変化
ユーザ行動数の推移

図 4.4: トップページ

レイヤ 1 2015 年 1 月 1 日 から 2015 年 1 月 7 日

レイヤ 2 2015 年 1 月 8 日 から 2015 年 1 月 14 日

レイヤ 3 2015 年 1 月 15 日 から 2015 年 1 月 21 日

またこの画面では、ノードのグループ化（グルーピング）を行うことができる。7 つのアクションに対して、そのアクションを行ったか行っていないかの 2 つのグループにノードを分けることができ、グルーピングを行った場合、それらのグループ間の流出入量を見ることができる。例えば、グルーピングのオプションとして、LIKE と CLIP を選択した場合は、ノードは以下の 5 種類に分類される。

1. LIKE も CLIP もしていないグループ
2. LIKE をしたが CLIP をしていないグループ
3. CLIP をしたが LIKE をしていないグループ
4. LIKE も CLIP もしたグループ
5. 新規登録ユーザのグループ

図 4.5 で示した画面は、レイヤ 1 を LIKE と CLIP でグループ化、レイヤ 2 を POST でグループ化した例になっている。

この画面は、Tunnel 株式会社が、各アクションをしたユーザとしていないユーザとを見比べて、ユーザがどのように推移したのかを観察する、全体俯瞰図としての役割を果たすことを目的としている。

4.5.3 Ranking 画面

図 4.6 に示す Ranking 画面は、ノードごとの 2 つの期間の変動率を表す画面であり、施策の効果を確認する個別比較図としての役割を果たすことを目的としている。ランキングの表形式で、ノードごとの人数、変動人数、変動率を表し、ランキング順のソートも可能である。期間は 1 つ目と 2 つ目を指定する。Flow 画面と同様のグルーピングも可能であり、グループごとに人数、変動人数、変動率を表示するようになる。また、登録日でユーザを絞り込むことができ、特定の範囲で登録されたユーザに関してのみ集計を行うこともできる。Ranking 画面に関しては、第 5 章で詳しく述べる。

4.5.4 Dashboard 画面

図 4.7 に示す Dashboard 画面は、アクションごとに、日付ごとのユーザ数に対するアクションを行ったユーザ数の割合を表示する画面である。登録日でユーザを絞り込むことができ、Ranking 画面同様、ある一定期間に行った施策の結果、アクションに関してどのように推移しているかを見ることができる画面である。

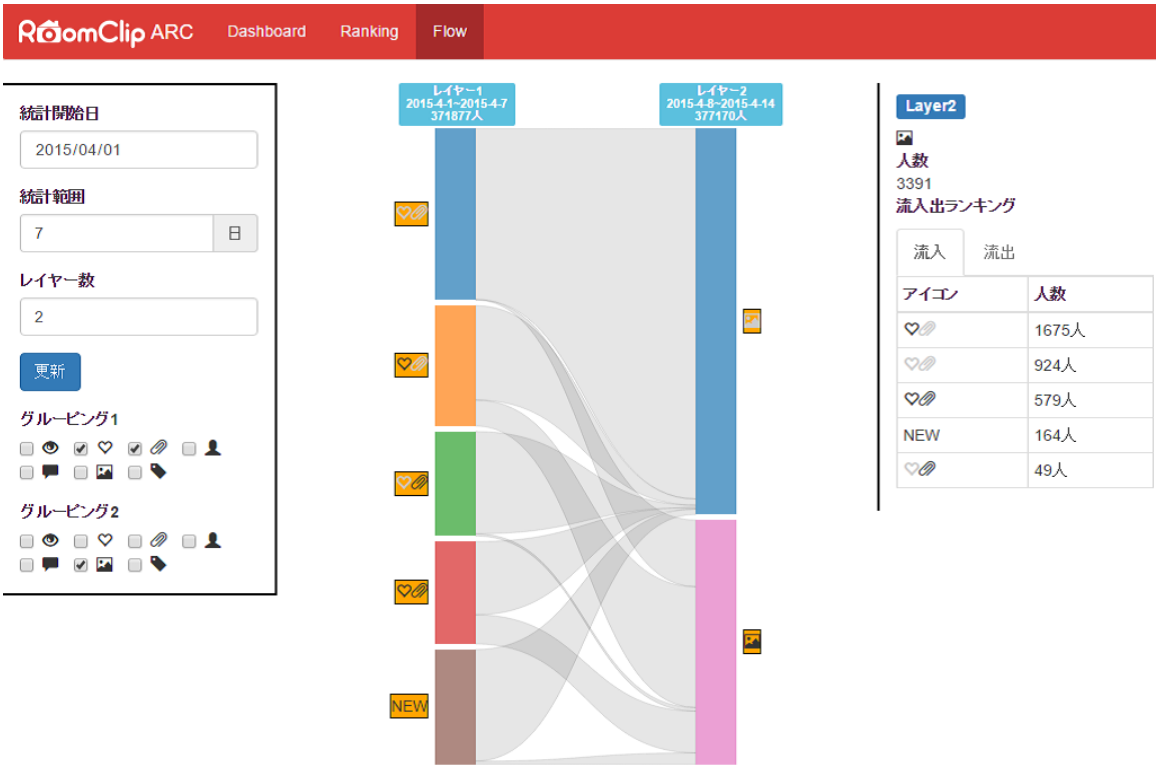

RoomClip ARC
Dashboard
Ranking
Flow
統計開始日
2015/04/01
統計範囲
7
日
レイヤー数
2
更新
グルーピング1
グルーピング2
レイヤー1
2015-4-1~2015-4-7
371877人
レイヤー2
2015-4-8~2015-4-14
377170人
NEW
Layer2
人数
3391
流入出ランキング
流入
流出
アイコン
人数
1675人
924人
579人
NEW
164人
49人

図 4.5: Flow 画面

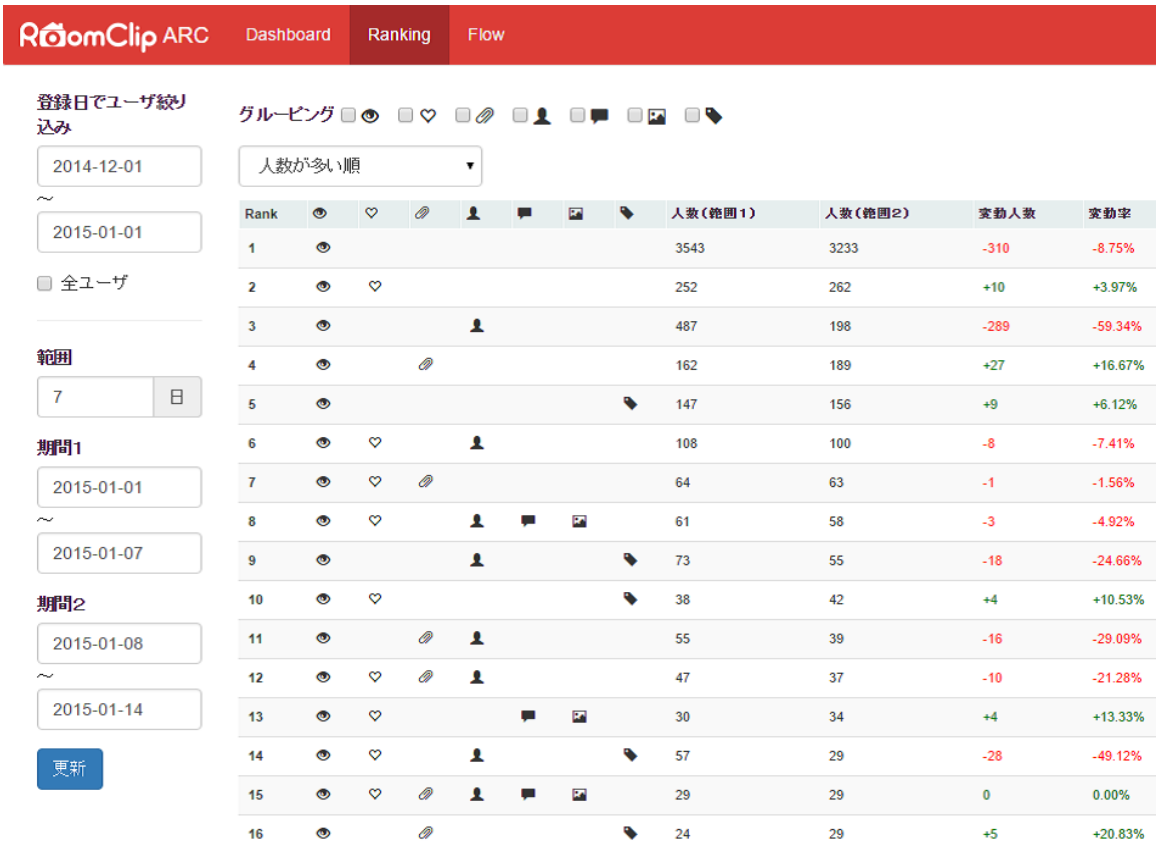


図 4.6: Ranking 画面

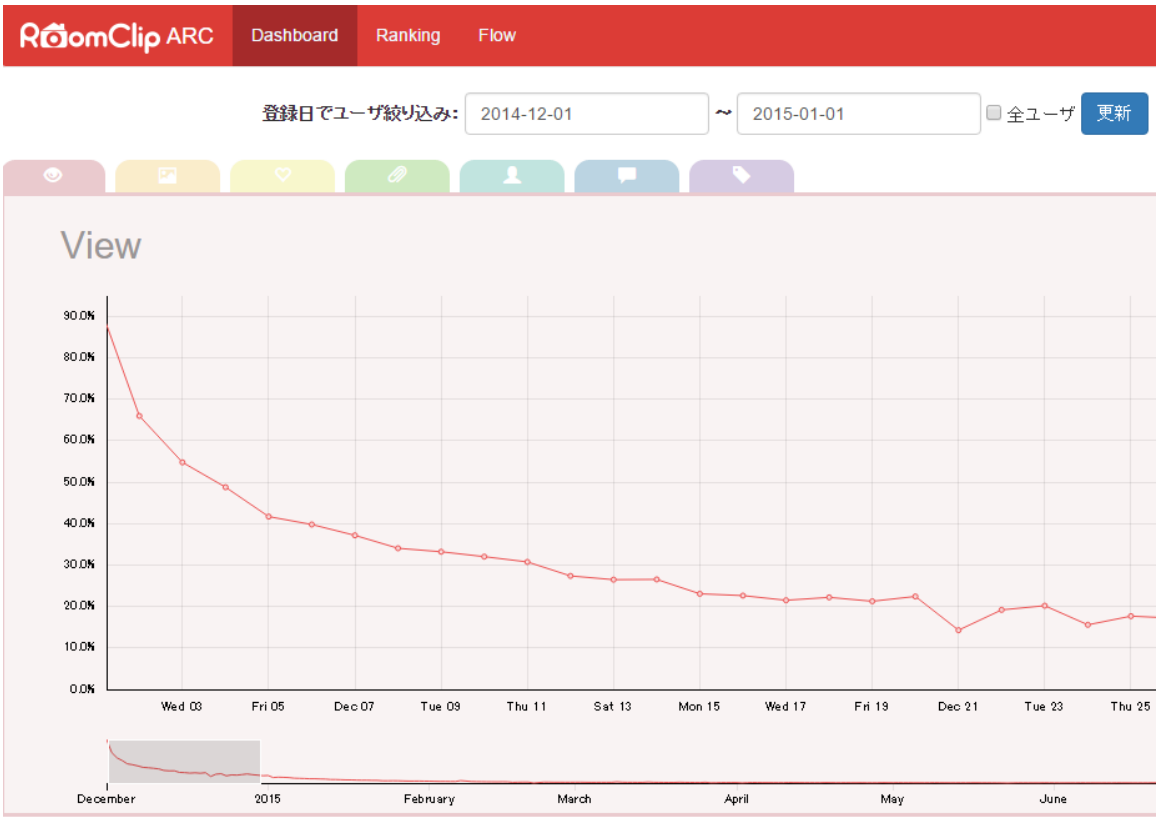


図 4.7: Dashboard 画面

第5章　Ranking画面の開発

ARC を開発するにあたり、筆者は 4.5.3 節で述べた Ranking 画面の設計・開発、フロントエンド・バックエンドサーバの構築およびバックエンドサーバ上の WebAPI の設計・開発を行った。本章では、Ranking 画面の意図、設計と実装および意図しているユーザシナリオの例を述べる。

5.1　Ranking画面について

ユーザの動向を把握することの 1 つのアプローチとして、何らかの施策を講じた前後のユーザアクティビティの変動を把握したり、キャンペーンなど特定の期間に登録したユーザアクティビティの変動を把握するということが挙げられる。何らかの施策の前後において、ユーザアクティビティの変化を定量的に把握することは、施策の効果を評価するための参考情報となり、次の施策を考えるために重要であると考えられる。しかし、現在 Tunnel 株式会社では、これらを把握することに苦慮している。

そこで ARC では、特定の 2 つの期間を期間 1・期間 2 としたとき、期間 1・期間 2 それぞれに関してノードごとの人数、期間 1 から 2 へのノードごとの増減人数（変動人数）、ノードごとの期間 1 の人数に対する期間 2 の割合（変動率）を見ることができる画面を開発した。ノード単位だけではなく、複数のノードをグループ化することもできる。グループ化はアクションに基づいて行い、各アクションに対して、そのアクションを行ったか行っていないかで分類していくことができるようにした。表示は表形式で、人数、変動人数、変動率でそれぞれランキング形式でソートすることができるため、ARC ではこの画面を Ranking 画面と呼ぶこととする。Ranking 画面の設計は、チーム全体、課題担当教員および Tunnel 株式会社の間で議論を重ね、Tunnel 株式会社の要望とチームからの提案を取り入れつつ、なるべく平易で使いやすいインタフェースを目指した。

5.2 節では、実際に ARC 上に実装した Ranking 画面のイメージとその使い方、5.3 節以降では、設計と実装および想定している具体的な利用例をユーザシナリオ例として示す。

5.2　画面イメージ

Ranking 画面へは、図 4.4 で示した ARC のトップページにアクセスし、左側の Ranking 画面のスクリーンショットをクリックするか、上部のナビゲーションバーから Ranking を選択することによって遷移する。Ranking 画面に遷移した時点の初期表示は図 4.6 に示されている。

ユーザの登録日の絞り込みは、左上の「登録日でユーザ絞り込み」の欄で行うことができる、日付入力欄の上部から下部までの日付は YYYY-MM-DD 形式で入力するが、クリックすることによって、カレンダー式入力欄が表示され、文字をキーボードでタイプしなくても日付を選択することができる（図 5.1）。また、全ユーザのチェックボックスを入れることにより、期間を指定せずに全ユーザに対して集計を行うことができる。

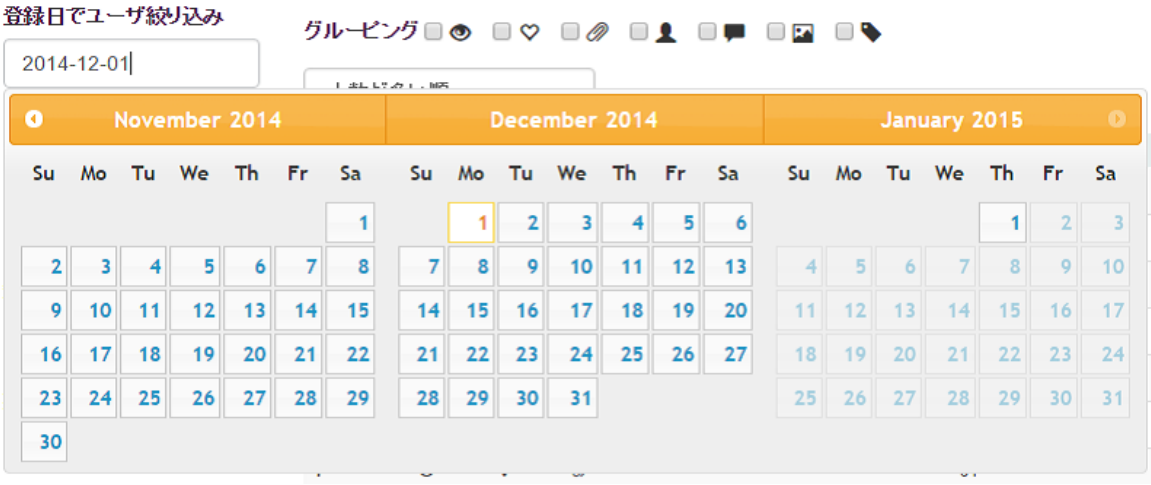


図 5.1: Ranking 画面：カレンダー式入力欄

集計の範囲は、左側の範囲、期間 1 および期間 2 に入力し、更新ボタンを押すことによって更新される（図 5.2）。期間 1 と期間 2 の日付指定欄は、ユーザ絞り込みの登録日指定欄と同様のカレンダー式入力欄で入力することができる。範囲欄は、期間 1 と 2 のそれぞれの期間の日数を表しており、範囲の日数を増やすことによって、自動的に期間 1 と期間 2 の期間終了日が伸びる。期間 2 の開始日は、期間 1 の終了日と重なってしまった場合、自動的に期間 1 の終了日の翌日に設定される。

グルーピングは画面上部のチェックボックスの列から行うことができる。それぞれのアイコンは glyphicons[*1] のアイコンであり、各アクションを表している。表 5.1 にアイコンとアクションの対応表を示す。グルーピングは各アクションのチェックボックスをオン・オフすることにより行い、チェックしたアクションをしたかしていないかでノードをグループ分けすることができる。

画面中央の表がランキングであり、各行が各ノードを表している。列は左からランク、アイコンが表 5.1 に示した各アクション、範囲 1 の人数、範囲 2 の人数、範囲 1 から 2 への変動人数、範囲 1 に対する範囲 2 の変動率を表している。このランキングは表の左上のセレクトボックスからソートを行うことができ、人数の多少、変動人数の多少、変動率の高低およびノード ID 順のそれぞれでソートすることができる（図 5.3）。

[*1] http://glyphicons.com/

図 5.2: Ranking 画面：範囲指定欄

表 5.1: アイコンとアクションの対応表

アイコン	対応するアクション名
	VIEW
	LIKE
	CLIP
	FOLLOW
	COMMENT
	POST
	FAVTAG

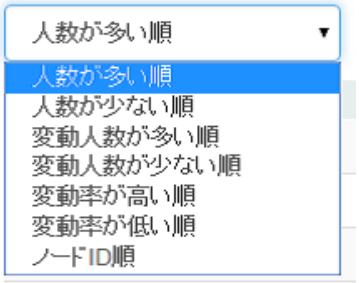


図 5.3: Ranking 画面：ソートのオプション一覧

またグルーピング時には、チェックしたアクションごとに 2 分割されるチェックしたアクションの集合を A とすると、チェックしたアクションの数は $|A|$ で、表は $2^{|A|}$ 通りの行数になる。しかし、VIEW を行っていない場合は他のアクションも行っていないという仮定から、VIEW が A の中に含まれている場合は、表は $2^{|A|-1}+1$ 通りの行数となる。例として、VIEW・LIKE・POST の 3 種類をチェックし、表に 5 通りのグループが見られる Ranking 画面の様子を図 5.4 に示す。

図 5.4: Ranking 画面：グルーピングを行った例

5.3 利用した技術

本節では、Ranking 画面を開発するにあたって利用した技術や実装の方法について述べる。

Ranking 画面は HTML5・CSS3・JavaScript によって構成されており、ARC のフロントエンドサーバで稼働している nginx による HTTP サーバによって提供される。

HTML および JavaScript では、jQuery[*2] を利用し、バックエンドサーバとの Ajax 通信を行う。また、シンプルなデータバインディングライブラリであり、jQuery との共存が比較的容易な Vue.js[*3] を利用し、コードの複雑さを減らすよう努めた。Vue.js の使用を決定した際の JavaScript ライブラリの比較一覧表は付録に添付する。

CSS としては、Twitter Bootstrap[*4] を利用し画面のデザインを行った。またカレンダー式入力欄として、jQuery UI[*5] を用いた。

[*2] https://jquery.com/
[*3] http://jp.vuejs.org/
[*4] http://getbootstrap.com/
[*5] https://jqueryui.com/

5.4　ユーザシナリオ例

本節では、Ranking 画面の実際の利用例を 2 つ挙げ、その利用例のシナリオを述べる。

クリスマスの投稿を促すキャンペーンの効果はあったのか？

Tunnel 株式会社が 2014 年のクリスマスまでの週 1 週間（12 月 19 日から 12 月 25 日）の間に、クリスマスに因んだインテリアの投稿を促すキャンペーンを行ったとき、その前の 1 週間（12 月 12 日から 12 月 18 日）とキャンペーン中の期間では写真投稿数の違いがあったかどうかを確認したいとする。この場合、Ranking 画面において以下のような処理フローで実現することができる。

1. 左側の範囲を 7 に設定
2. 期間 1 を 2014-12-12 に設定
3. 期間 2 を 2014-12-19 に設定
4. 左上の「全ユーザ」をチェック
5. 更新ボタンを押す
6. グルーピング欄の POST にチェックを入れる

以上のフローで、表には写真投稿をしたグループとしていないグループの比較が表示される。ここで期間 2 で写真投稿をしたグループの人数が増えていた場合キャンペーンは成功、減っていた場合はキャンペーンは失敗と捉えることができる。

クリスマスキャンペーンで登録したユーザは、その後も継続して利用しているか？

Tunnel 株式会社が 2014 年のクリスマスまでの 1ヶ月（11 月 26 日から 12 月 25 日）の間に、クリスマスに因んだインテリアの投稿を促すキャンペーンを行ったとき、キャンペーン後の 1ヶ月（12 月 26 日から 1 月 24 日）とまた次の 1ヶ月（1 月 25 日から 2 月 23 日）で継続して利用しているかどうかを確認したいとする。この場合、Ranking 画面において以下のような処理フローで実現することができる。

1. 登録したユーザの範囲を 2014-11-26 〜 2014-12-25 に設定
2. 左側の範囲を 30 に設定
3. 期間 1 を 2014-12-26 に設定
4. 期間 2 を 2015-01-25 に設定
5. 更新ボタンを押す

以上のフローから、全体を見渡してみて、全体的に人数が減っているようなら、継続して利用しているユーザが減っており、増えているようなら継続して利用しているユーザが増えていることが推測できる。ここでも、各アクションごとにどのようなユーザが増減しているかを確認することができる。

5.5 フィードバックと考察

開発した Ranking 画面は、実際に Tunnel 株式会社に使っていただき、使用感やどのような操作をしてどう思ったかなどのフィードバックを頂いた。使っていただいた期間は約 1 週間で、実際に ARC にアクセスできる URL を渡し、実際に Tunnel 株式会社からアクセスしてもらい利用してもらった。サーバには BASIC 認証がかけられており、URL だけではなく、アクセスするための ID とパスワードも同時に配布した。その結果、以下のようなフィードバックを頂いた。以下は Tunnel 株式会社から頂いたフィードバックの文書を抜粋・整形したものである。

どのような操作をし、どう思ったか

- 単純にどういう行動している人がいっぱいいるのかを確認した
  - 今まで「いいね」何人、「クリップ」何人という単独でみていたが、それがベン図のように見えたため、「主にどういう行動をユーザーがしているのか」が見渡せた
  - ただし相関がみれるわけではないので、それ以上の分析・インサイトはなかった
- ある日の登録者を絞り込み、登録日から 1 日毎に期間をずらして変動率をおっかけた
  - 初日からアクティビティは減る一方であることがわかった
  - あるタイミングでアクティビティが増えるという状況がもしあるならば、例えば「本当は撮影するユーザーでも、10 個くらいいいねしたりコメントしたり様子を見たあとに投稿する」ようなことが言えるかもしれない、と考えた。
- ある期間に登録したユーザーのアクティビティがどう減少していくのかをデイリー単位でみようとしたが、できなかった
  - きっと減少の速度がアクティビティによって変わるはずだ、という想定があった
- どの行為をやっているユーザーが最後までのこる（ゼロにならない）のかをみた
  - スティッキーに使っているユーザーがどのアクティビティをやっているのかを確認したかったが、結構変動が激しく、捉えられなかった
- ランキングの推移を確認しようとした
  - ある時に登録した人のランキング変位がどのくらいのタイミングでおこるのかを見ようとしたが、煩雑でできなかった

総評

「なにもない状態で分析しろ」といわれたら、やはりフローについて見たがる傾向にあると自己認識した。ただし、もし「いいね促進キャンペーン」などを特定の期間ではっていた場合、どの程度変動するのかを期間登録者ベースでみれると、効果測定としてはとてもよいと感じた。結果、ランキング機能は「標本の絞り込み」と「期間」が非常に重要で、つまり「標本を絞り込んだ理由」か「期間を絞り込んだ理由」が分析の要になると思った。具体的には下記 2 つ。

- ある期間にキャンペーンを張った場合、それが効果があったのか？の確認
- ある特殊な経路で入ってきた登録者の場合、それがどのように変遷したのか?の確認

のようなことに効果的だと言えそうだ。その場合これは「どういうユーザー特性があるのか」といったような、現状分析にはあまり向いておらず、施策の評価として意味をもつ、ということなので、なにも施策をうっていない状態での仮説立脚には至らなかった。

以上のフィードバックから、開発チームの意図通り、特定の期間に行ったキャンペーンに関するユーザの推移を見るために ARC を利用できることがわかり、ARC の Ranking 画面の有用性は確認することができた。

しかし、Ranking 画面は施策や仮説が立っていない状態で分析するためには向かないことが再確認できた。また、1 日単位などの細かい範囲での分析も、現在のインタフェースでは範囲の指定が煩雑で難しいことがわかった。これは、あくまで Ranking 画面は 2 つの期間の間のみでノードごとに増減などの集計を行っており、ノード間の遷移や複数期間の時系列の推移を見ることができず、向いていないためであると考えられる。よって、各ノードがどのように変遷したのかを見たい場合は、ノード間の流出入を見ることができる Flow 画面を利用する必要がある。

また、ユーザアクティビティがどのような条件で増えるのかなどの、ユーザアクティビティに関するパターンはこの画面では発見することができないこともわかった。

したがって、細かい単位の期間で集計を見たい場合、また 3 つ以上の期間で集計を見たい場合のインタフェースを考案することが、Ranking 画面としての今後の課題と考えられる。

第6章　WebAPIの開発

筆者は、利用データを JSON 形式のデータで提供する WebAPI の設計・実装を行った。本章では、WebAPI の概要、設計と実装およびその工夫点について述べる。

6.1　概要

ARC では、1 つのサーバ上に Web サービスとしてすべてを実装するモノリシックなアーキテクチャではなく、Web ページを提供する部分、データベースからデータを取得し加工する部分およびデータベースそのものを分離することにした。ここで Web ページを提供する部分をフロントエンド、データを取得し加工する部分をバックエンドと呼ぶことにする。このように 3 つのサブシステムに分割したのは、ARC を開発するにあたって、担当者をフロントエンドとバックエンドに分けることによって効率化を図ろうとした意図がある。

データの取得と加工を行うバックエンドでは、データベースからのデータの取得は SQL で行い、Python のフレームワークである Pyramid を利用し RESTful な WebAPI として提供する。フロントエンドからバックエンドサーバに、URL を指定し HTTP リクエストを送信すると、処理結果を JSON 形式で提供するようにしている。本章では、その WebAPI の開発に関して述べる。

6.2　サーバ構築

筆者は、ARC で使用するフロントエンドサーバとバックエンドサーバの構築･設定を行った。2 つのサーバはどちらも AWS の EC2 のインスタンスを利用し、OS は Amazon Linux 2015.03 を利用している。それぞれのサーバで開発メンバー全員分の公開鍵を登録し、その公開鍵を用いて SSH で接続できるようにした。また、どちらも BASIC 認証を設定し、意図しない第三者からのアクセスを防ぐようにした。フロントエンドサーバは、nginx をインストールし、それを HTTP サーバとして利用できるようにした。次節では、特にバックエンドサーバに関し、利用したフレームワークの詳細を述べる。

6.3　設計と実装

本節では、WebAPI の設計と実装に関する詳細を述べる。

6.3.1 利用したフレームワーク

WebAPI の提供には、Python を用いた Web フレームワークであり、軽量かつ堅牢な Pyramid を利用した。Pyramid は 1 ファイルで Web アプリケーションを作成できるなど、プロジェクトの作成が容易であり、また簡潔な記述でルーティングを行うことがでるため、WebAPI の実装に向いていると同時に、Python の豊富なライブラリを活用できるというメリットがある。

6.3.2 実装

本節では、ARC の WebAPI の実装方法と工夫点について述べる。ARC の WebAPI は、各画面の JavaScript から jQuery の Ajax 機能を利用して呼び出され、結果は JSON 形式（application/json）で返される。例として、ARC の Ranking 画面を開いたときの処理フローを図 6.1 に示す。

WebAPI の実装には、Pyramid のルーティング機能を用いて、Pyramid の main 関数内の configuration 部分で各 API の URL に対しルーティングを設定し、ルーティング先では、データベースへの接続を行いデータを取得し JSON に整形する関数を呼び出し、それを response として返すことによって実現している。データベースへの接続は、Python のパッケージである psycopg2[*1] で、PostgreSQL を利用した接続を行っている。また、データベースへの接続認証の部分では、Pit パッケージ [*2] を利用し、パスワードやデータベース名などはソースコード内に埋め込まないようにした。また、実装は RESTful な API[7] とし、ディレクトリ構造のような URL 設計を行い提供することにした。実装にはオライリー・ジャパンの RESTful Web Services[8] や、Web 上のブログなどを参考にした。例として Ranking の API を挙げ、Pyramid 上の処理の処理の概念図を図 6.2 に示す。

また、ARC で使う API だけではなく、機械学習の API も同時にバックエンドサーバから提供することとした。本プロジェクトでは、開発チームメンバーである河越によって RoomClip の利用データに対する機械学習モデルが開発されたため、API としては、開発した機械学習モデルによる「やめそうなユーザ」の予測結果と、探索的因子分析の結果を提供する。これは、Tunnel 株式会社が直接 WebAPI として利用するための提供である。

Python のバージョンは 3.5.0 である。機械学習の API も提供するため、Anaconda 2.4.0 [*3] を利用し数値計算系ライブラリを一括インストールしている。表 6.1 で、直接ソースコード内で指定して利用したパッケージの一覧を示す。依存性などで内部的に利用するパッケージは考慮しない。

[*1] http://initd.org/psycopg/docs/
[*2] https://pypi.python.org/pypi/pit/
[*3] https://www.continuum.io/

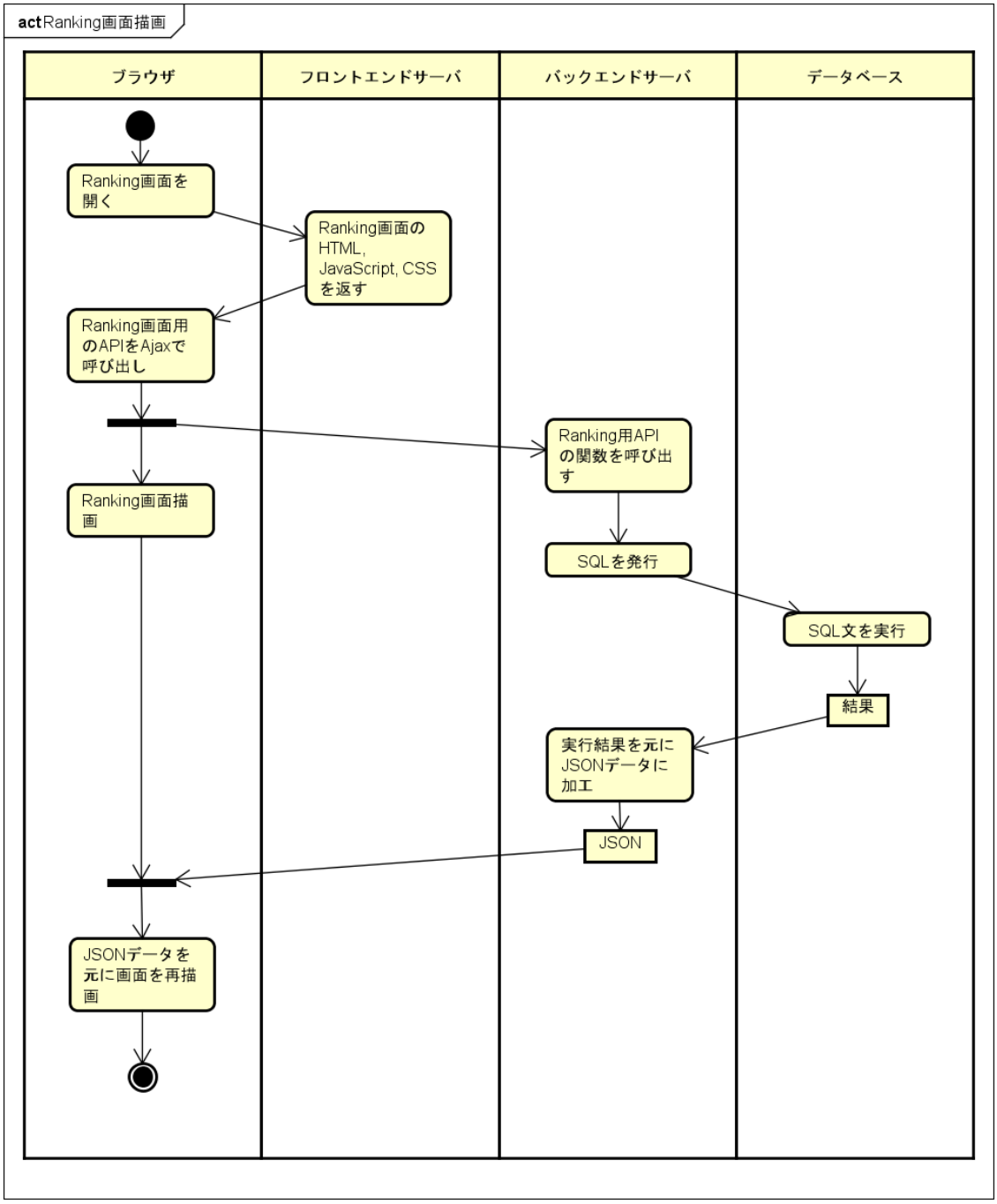


図 6.1: Ranking 画面を開いた時の処理フロー

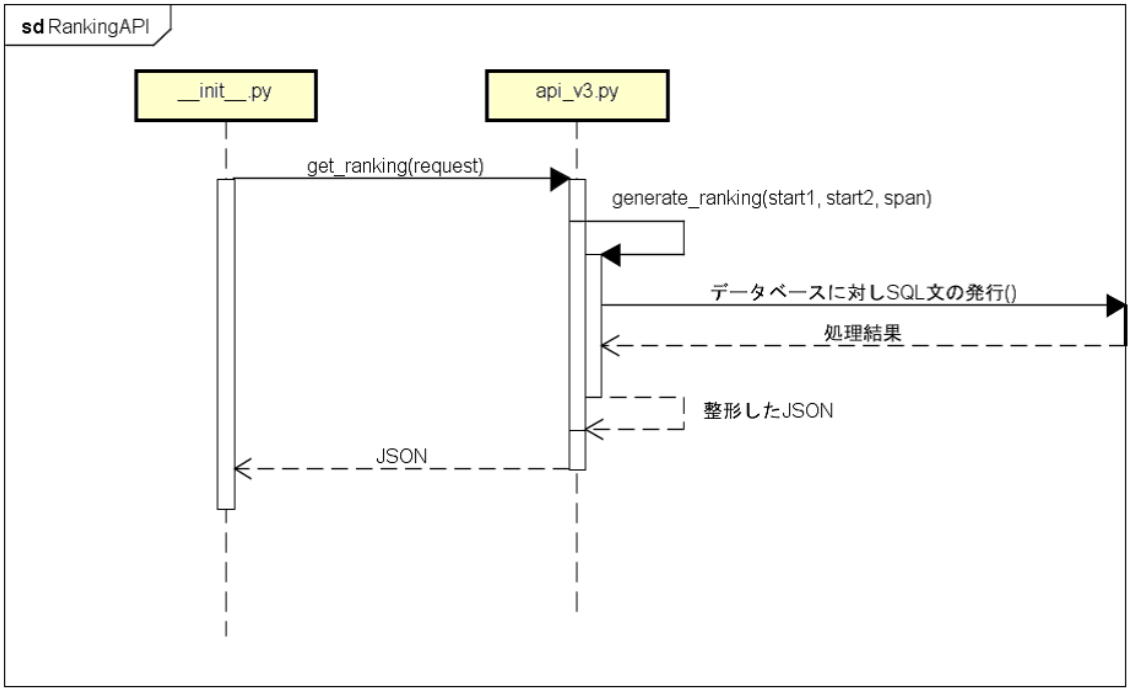


図 6.2: RankingAPI の処理の概念図

表 6.1: 利用した Python パッケージの一覧

名前	バージョン
pyramid	1.5.7
pit	0.1.3
psycopg2	2.6.1
numpy	1.10.1
scipy	0.16.0
scikit-learn	0.16.1
PypeR	1.1.2
matplotlib	1.4.3

6.3.3 工夫点

工夫点としては、Pyramid 上の Python で計算量の多いデータ処理を行うのではなく、SQL 文でデータ処理の大部分を行い、AWS の Redshift（開発中は RDS）の性能を引き出し、Pyramid 上では、計算量の少ない JSON ファイルの整形だけを行うようにしたことが挙げられる。特に Redshift は列指向データベースであり、COUNT などの SQL の集約関数の実行速度が非常に早い。下に示すのは、Ranking 画面の API で利用する、user_activity テーブルから node テーブルへ JOIN するまでの SQL を、COUNT を使って得たい結果を直接取得するものと、COUNT を使う前までの結果を取得するものを、Redshift に PostgreSQL で接続して EXPLAIN を使い、Query Plan と実行時間を比較した結果である。なお、Query Plan は、出力結果そのままではなく、ページに載せるために一部整形を施してある。この結果から、COUNT を使っているほうは Query Plan が効率化され実行時間が早くなっていることが実際に確認することができる。

COUNT を使って得たい結果を直接得る SQL 文を計測したもの（COUNT あり）を次に示す。

```
test=# EXPLAIN
SELECT
    COUNT(*), views, likes, clips, follows, comments, posts
FROM
    (
        SELECT
            u_id, SIGN(SUM(u_view)) AS views,
            SIGN(SUM(u_like)) AS likes,
            SIGN(SUM(u_clip)) AS clips,
            SIGN(SUM(u_follow)) AS follows,
            SIGN(SUM(u_comment)) AS comments,
            SIGN(SUM(u_post)) AS posts
        FROM
            user_activity
        WHERE
            u_date >= '2015-08-01'
        AND u_date <= '2015-08-07'
        GROUP BY
            u_id
        ORDER BY
            u_id
    ) s
GROUP BY
    views, likes, clips, follows, comments, posts
;
                            QUERY PLAN
------------------------------------------------------------------------
 XN HashAggregate  (cost=107280.89..107318.07 rows=14870 width=48)
   ->  XN Subquery Scan derived_table1  (cost=96500.72..104678.78
                                         rows=148692 width=48)
      ->  XN HashAggregate  (cost=96500.72..103191.86
                             rows=148692 width=16)
         ->  XN Seq Scan on user_activity  (cost=0.00..44538.79
                                           rows=2969253 width=16)
            Filter: ((u_date <= '2015-08-07'::date) AND
                     (u_date >= '2015-08-01'::date))
```

COUNT を使う前までの結果を取得する SQL 文を計測したもの（COUNT なし）を次に示す。

```
test=# EXPLAIN
SELECT
    u_id, SUM(u_view) AS views, SUM(u_like) AS likes,
    SUM(u_clip) AS clips, SUM(u_follow) AS follows,
    SUM(u_comment) AS comments, SUM(u_post) AS posts
FROM
    user_activity
WHERE
    u_date >= '2015-08-01'
AND u_date <= '2015-08-07'
GROUP BY
    u_id
ORDER BY
    u_id
;
                            QUERY PLAN
------------------------------------------------------------------------
  XN Merge  (cost=1000000092751.83..1000000093061.59
             rows=123902 width=16)
     Merge Key: u_id
     ->  XN Network  (cost=1000000092751.83..1000000093061.59
                      rows=123902 width=16)
        Send to leader
        ->  XN Sort  (cost=1000000092751.83..1000000093061.59
                      rows=123902 width=16)
           Sort Key: u_id
           ->  XN HashAggregate  (cost=80411.91..82270.44
                                  rows=123902 width=16)
              ->  XN Seq Scan on user_activity  (cost=0.00..37113.19
                                                 rows=2474213 width=16)
              Filter: ((u_date <= '2015-08-07'::date) AND
                       (u_date >= '2015-08-01'::date))
```

また、COUNT ありと COUNT なしを AWS のコンソール上で動作させ、所要時間を可視化した。COUNT ありを図 6.3、COUNT なしを図 6.4 に示す。COUNT なしでは、テーブルのソートからマージをシーケンシャルに行っているのに対し、COUNT ありでは HashAggregate を利用し並列処理を行っていることがわかる。

またもう一つの工夫点として、API のバージョン分けを行った。開発を進めるにあたって、

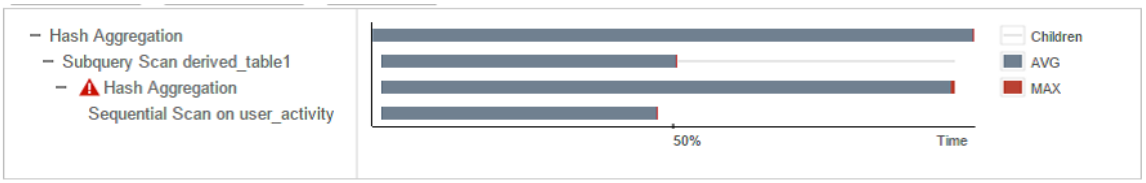


図 6.3: Query Plan：COUNT あり

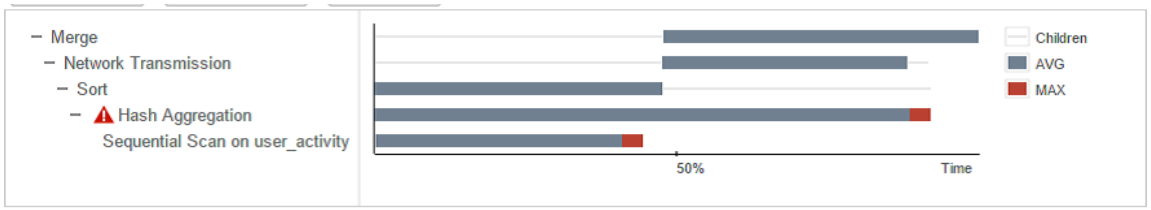


図 6.4: Query Plan：COUNT なし

画面の仕様変更から波及し API の仕様変更を行う必要が頻繁にあり、仕様変更前と仕様変更後の両方の API を使用する必要がある局面があったため、各 API の URL の始めに`/api/v1` など、接頭辞としてバージョン番号をつけ、仕様変更前と仕様変更後のどちらのバージョンの API も同時に使えるようにした。

6.4　提供する API

本節では、最終的に提供することになった API に関して述べる。API の最終バージョンは v3 であり、URL の接頭辞として`/api/v3` が付く。URL 内の {} 内で囲まれている部分はパラメタを表し、説明文中にある形式の文字列で適宜置き換えて使用する。各バージョン内で使用している SQL 文は、本報告書の付録として添付する。以下に、提供している API の仕様を示す。これらの仕様は、仕様書としてドキュメントにまとめてチーム内で共有されている。

GET /api/v3/ranking/{start1}/{start2}/{span}

集計したランキングを返す。
start1：起点となる日付 1 (YYYY-MM-DD)
start2：起点となる日付 2 (YYYY-MM-DD)
span：集計期間（起点となる日付を含める）

GET /api/v3/ranking/{start1}/{start2}/{span}/registration/{startreg}/{endreg}

指定した期間に新規登録したユーザに関して集計したランキングを返す。
startreg：新規登録期間開始日

endreg：新規登録期間終了日
start1：起点となる日付 1 (YYYY-MM-DD)
start2：起点となる日付 2 (YYYY-MM-DD)
span：集計期間（起点となる日付を含める）

GET /api/v3/flow/{start1}/{start2}/{span}
集計したノード間流出入を返す。
start1：起点となる日付 1 (YYYY-MM-DD)
start2：起点となる日付 2 (YYYY-MM-DD)
span：集計期間（起点となる日付を含める）

GET /api/v3/flow/{start1}/{start2}/{span}/all
集計したノード間流出入を返す。＜ NodeID65(noview), 66(new) 含む＞
start1：起点となる日付 1 (YYYY-MM-DD)
start2：起点となる日付 2 (YYYY-MM-DD)
span：集計期間（起点となる日付を含める）

GET /api/v3/transition/actions
全ユーザを対象とし、全期間の日付ごとに全ユーザ数と各アクションを行った人数を返す。

GET /api/v3/transition/actions/{startreg}/{endreg}
startreg から endreg の間に登録したユーザを対象とし、全期間の日付ごとに全ユーザ数と各アクションを行った人数を返す。
startreg：ユーザの登録範囲開始日
endreg：ユーザの登録範囲終了日

GET /api/v3/transition/actions/{startreg}/{endreg}/{start}/{end}
startreg から endreg の間に登録したユーザを対象とし、start から end までの期間の日付ごとにユーザ数と各アクションを行った人数を返す。
startreg：ユーザの登録範囲開始日
endreg：ユーザの登録範囲終了日
start：集計する範囲の開始日
end：集計する範囲の終了日

GET /api/v3/transition/registrations
全期間の日付ごとの登録者の推移を返す。

GET /api/v3/transition/registrations/{start}/{end}
start から end までの日付ごとの登録者の推移を返す。start：範囲の開始日
end：範囲の終了日

返される JSON のフォーマットは以下のように指定した。大カッコ内の数字は、示しているオブジェクトがその分だけ繰り返され配列となることを示している。また、transition の data は、データが続く限りの配列となっている。

◆ ranking の JSON フォーマット

```
{
  start1 : 2015-07-20, // 起点となる日付 1
  start2 : 2015-07-27, // 起点となる日付 2
  span : 7, // start1,2 からの集計期間 (start1,2 も含める)
  ranking[65] :  [ {
    nodeid : 4, // NodeID
    actions : [“ VIEW ”, “ LIKE ”, “ POST ”], // Actions
    range1 : 2000, // start1 に関する集計の結果
    range2 : 2200 // start2 に関する集計の結果
  } ]
}
```

◆ flow の JSON フォーマット

```
{
  start1 : 2015-07-20, // 起点となる日付
  start2 : 2015-07-27, // 起点となる日付 2
  span1 : 7, // start1,2 からの集計期間 (start1,2 も含める)
  nodes[65] :  [ {
   nodeid : 4, // NodeID
    actions : [“ VIEW ”, “ LIKE ”, “ POST ”], // Actions
    range1 : 2000, // start1 に関する集計の結果
    range2 : 2200 // start2 に関する集計の結果
    flowout[65] : { // 各ノードに対して出ていく数に関する情報
      nodeid : 2, // NodeID
      actions : [“ VIEW ”, “ COMMENT ”], //Actions
      num : 100 // 出ていく数
    }]
}
```

◆ flow/all の JSON フォーマット

```
{
  start1 : 2015-07-20, // 起点となる日付
  start2 : 2015-07-27, // 起点となる日付 2
  span1 : 7, // start1,2 からの集計期間 (start1,2 も含める)
  nodes[66] :  [ {
   nodeid : 4, // NodeID
```

```
    actions : [“ VIEW ”, “ LIKE ”, “ POST ”], // Actions
    range1 : 2000, // start1 に関する集計の結果
    range2 : 2200 // start2 に関する集計の結果
    flowout[66] : { // 各ノードに対して出ていく数に関する情報
      nodeid : 2, // NodeID
      actions : [“ VIEW ”, “ COMMENT ”], //Actions
      num : 100 // 出ていく数
    }]
}
```

◆ transition/actions の JSON フォーマット

```
{
  start: 2014-01-01, // 開始日
  end: 2015-01-01, // 終了日
  startreg: 2014-12-01 // 登録開始日
  endreg: 2014-12-01 // 登録終了日
  data : [{
    date: “ 2015-04-01 ”, // 日付
    views: “ 50 ” // 以下、この日に各アクションを行ったユーザ数
    likes: “ 50 ”
    clips: “ 50 ”
    follows: “ 50 ”
    comments: “ 50 ”
    posts: “ 50 ”
    favtags: “ 50 ”
  }]
}
```

◆ transition/registration の JSON フォーマット

```
{
  start: 2014-01-01, // 開始日
  end: 2015-01-01, // 終了日
  data : [{
    date: “ 2015-04-01 ”, // 日付
    registrations: “ 100 ” // 登録人数
  }]
}
```

機械学習の API 仕様は、内部資料ではなく Tunnel 株式会社に提出する資料となるため、API ドキュメンテーションのクラウドサービスである apiary[*4] を利用し作成した。ここでは、apiary

[*4] https://apiary.io/

で採用されている API Blueprint Syntax による Markdown 方式のドキュメントを示す。

```
FORMAT: 1A

# ARC-ML

Analytics for RoomClip (ARC) の機械学習 WebAPI のドキュメントです。

# Random Forest を用いた機械学習による予測 [/predict_rf]

### 予測結果を取得する [GET /predict_rf/{start}]
```

指定した日付から 1 週間（7 日間）に何らかのアクションを起こしたユーザのうち、次の 7 日間で何もアクションを起こさないユーザ（ノンアクティブユーザ）を予測し、そのユーザ ID の一覧を返します。

予測には、あらかじめ Random Forest で学習させてあるモデルを用います。

```
+ Parameters
    + start (string) - この日から 7 日間にアクションを行ったユーザに対して予測をする（YYYY-MM-DD 形式）

+ Response 200 (application/json)

        {
            "start": "2014-12-01",
            "end": "2014-12-07",
            "non_active_users": [10, 200, 3000]
        }



# 探索的因子分析 [/efa]

### 1 週間の分析結果を取得する [GET /efa/{start}]
```

指定した日付から 1 週間（7 日間）で各ユーザが行った 7 つのアクションそれぞれの回数を集計し、それらを変数として探索的因子分析を行います。

分析には R を使用し、返される結果（Response の result）は PypeR によるレスポンスを JSON serialize したものです。

```
+ Parameters
    + start (string) - この日から 1 週間の分析を行う（YYYY-MM-DD 形式）

+ Response 200 (application/json)

        {
            "start": "2014-12-01",
            "end": "2014-12-07",
            "result": {
-------- PypeR による JSON レスポンスと全く同じ形式のため省略 --------
            }
        }
```

数週間の分析結果を取得する [GET /efa/{start}/{layers}]

集計する週間の数を layers として指定できるバージョンです。
返される 1 週間のものと同じです。例は 3 週間の場合です。

```
+ Parameters
    + start (string) - この日から数週間の分析を行う（YYYY-MM-DD 形式）
    + layers (int) - 週間数の指定

+ Response 200 (application/json)

        {
            "start": "2014-12-01",
            "end": "2014-12-21",
            "result": {
-------- PypeR による JSON レスポンスと全く同じ形式のため省略 --------
            }
        }
```

第7章　プロジェクトマネジメント

本章では、本プロジェクトのマネジメントについて述べる。

7.1　プロジェクトの進め方

本プロジェクトは、筆者をリーダとした4人のチームで開発を行った。開発中は1週間に1回程度、課題担当教員を交えてミーティングを行ったほか、技術的な問題やチーム全体で共有するべき情報があるときなどは、1週間に1回から2回程度、チームミーティングと称しチーム内でもミーティングを行った。それ以外の時間はチーム全体のコアタイムは設けず、各自で作業を行い、何らかの問題が発生したときは、後述のチャットツールやメールを用いて報告し、解決できるメンバーがサポートやアドバイスなどを行った。筆者は開発チームリーダとして、ミーティングのリマインダの発行やTunnel株式会社出張時に事務担当者へ事務手続きの連絡などを行った。

Tunnel株式会社へ7月9日・10月6日・11月27日の3回赴き、進捗報告と現状のARCのデモを行った。そのときの様子を図7.1に示す。このデモでは、実際にARCを使っていただき、どのように操作しているか、どのような分析に利用するかといったユーザテストを兼ねたデモとした。

全体のスケジュールとしては、ウォーターフォールモデルのような全体的な開発期間を設けずに、チームによる開発と課題担当教員によるフィードバックを約1週間単位で繰り返した形となった。実績として、おおまかには、4月から6月までは既存のツールの調査とARCで解決したいTunnel株式会社の現状の問題の洗い出し、7月から11月まではARCの設計・開発、12月はTunnel社からのフィードバックを受けたARCの修正と特定課題研究報告書の執筆を行った。

7.2　役割分担

本プロジェクトの開発チームの役割分担は、開発する機能・画面が確定した時点で行った。分担は各機能の開発担当範囲と責任範囲を表し、開発を進めていく上で発生した技術的な障害などは、チーム内または課題担当教員と相談しながら解決した。役割分担を表7.1に示す。

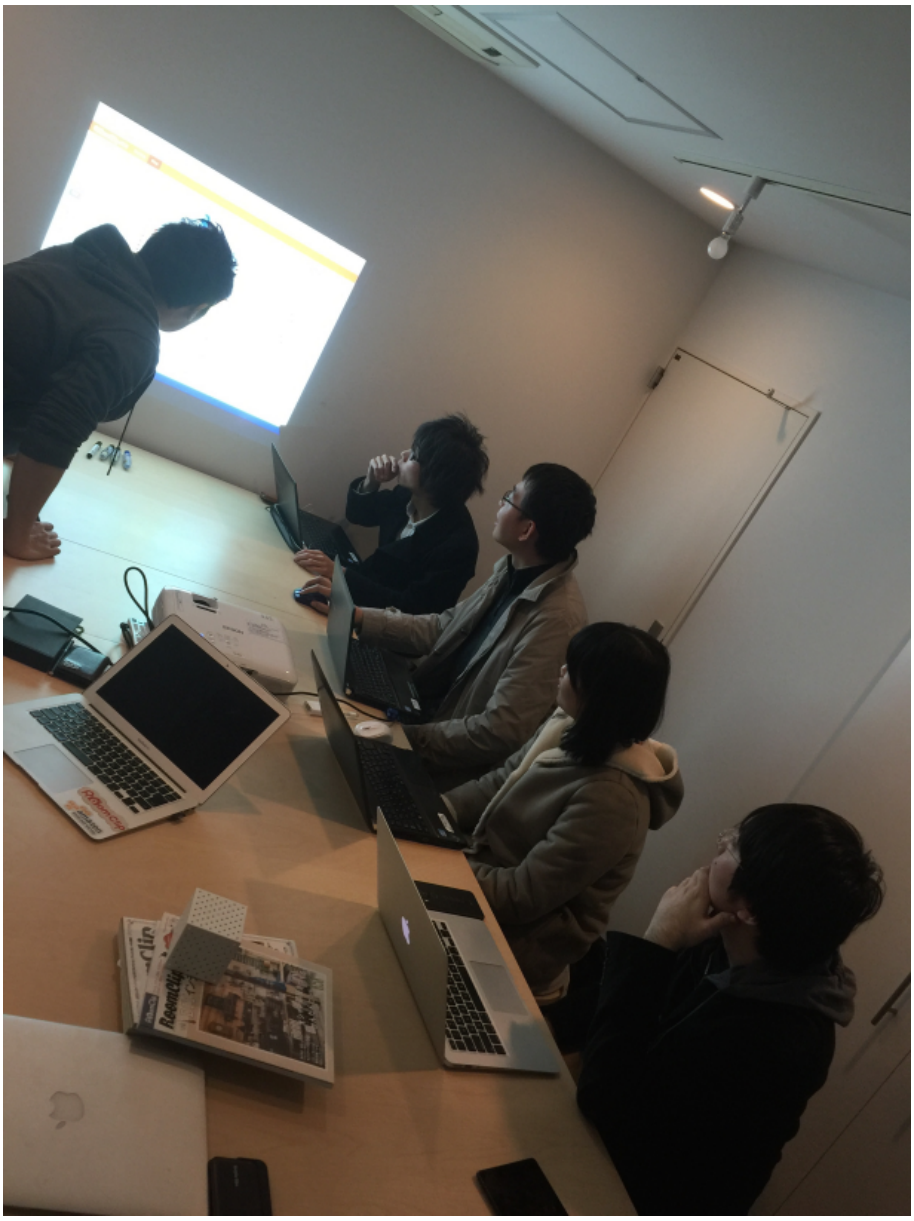


図 7.1: Tunnel 社との打ち合わせの様子

表 7.1: 役割分担表

メンバー名	担当
河越嵩介	データベースの設計・実装 機械学習モデルの実装
古谷翔太	Ranking 画面の設計・開発 サーバの構築 WebAPI の構築
崔野	Flow 画面の設計・実装
万シャンシャン	トップページの設計・実装 Dashboard 画面の設計・実装

7.3 利用したツール

本プロジェクトでは、開発の効率化を図るため、さまざまなクラウドサービスを利用した。これらのツールは筆者が提案し導入を行ったもので、開発と同時並行で、これらのツールの管理や、他チームメンバーへのツール使用法などの助言を行った。本節では、本プロジェクトで利用したツールについて述べる。

バージョン管理

ARC のバージョン管理には、GitHub[*1] を利用した。GitHub Education[*2] によって、指定したメンバのみが利用できるプライベートリポジトリを無料で作成できるようにし、本プロジェクト用の団体アカウントを取得し、管理をおこなった。フロントエンド側とバックエンド側のリポジトリは分割し、それぞれ別々に開発できるようにした。

タスク管理

チームメンバーが現在どのような作業を行っているのかを可視化するため、Trello[*3] を利用した。Trello では、スクラムに代表されるアジャイルソフトウェア開発で利用されているカンバンのように、Todo・Doing・Done のリストを作成し、さらに 3 つのリストを付け加え、それぞれのタスク、それを担当するメンバーおよびそのタスクの状態を明確にした。表 7.2 にリストの一覧を示す。タスクを作成した時にはまず Todo のリストにタスクを追加する。着手した際に Doing に移し、完了したら Check に移す。ミーティング時にフィードバックを受け、完了したら Done に移す。ミーティング時に議論が完了しなかったタスクは、Pending に移し、不要と判断されたタスクは Old に移すというワークフローで開発を行った。

スケジュール管理

[*1] https://github.com/
[*2] https://education.github.com/
[*3] https://trello.com/

スケジュールの管理には Google カレンダー [*4] をメンバーで共有し利用した。カレンダーには主に予定したミーティングの日時や研究開発プロジェクトにおける予定を記入した。

チャットツール

チーム内の連絡や課題担当教員との連絡は、メールだけではなく Slack[*5] を利用した。Slack は会話を複数のチャンネルに分けることができるチャットツールであり、チーム内での連絡・議論のチャンネルや、課題担当教員を交えた連絡では別々のチャンネルを利用した。

表 7.2: Trello のリストの一覧

名前	意味
Todo	やるべきタスク
Doing	現在着手しているタスク
Check	フィードバック待ちのタスク
Done	完了したタスク
Pending	議論が完了しておらず保留のタスク
Old	議論の結果不要となったタスク

[*4] https://calendar.google.com/
[*5] https://slack.com/

第8章　おわりに

本プロジェクトでは、RoomClip の利用データを用いて、ユーザの動向を把握することができるツールを開発した。

RoomClip を開発・運営している Tunnel 株式会社は、RoomClip をさらに発展させるため、さまざまな施策を講じているが、その施策の効果の確認や施策の対象とするユーザの選定に苦慮している。この問題を解決するため、本プロジェクトでは、ユーザの現状や動向を把握し、施策の考案や講じた施策の効果の確認を行うことができるツール「ARC」を開発した。

開発にあたっては、現在利用されている SNS の分析ツールの調査を行った。結果として、SNS の分析ツールには大きくわけて汎用データ分析ツールと特化型分析ツールの 2 種類が存在することがわかり、これらの 2 種類の特徴やユーザインタフェースを ARC の開発の参考にした。

筆者は、ユーザのグループを行動の種類別に分け、ある 2 つの期間でそれらのグループで人数がどのように変化したかを見る Ranking 画面と、ARC で利用するユーザデータを WebAPI の形で提供する API の開発を行った。WebAPI では、本プロジェクトで平行して開発した機械学習モデルによるユーザ動向の予測結果を提供する部分も同時に作り、Tunnel 株式会社に提供できるようにした。また、ARC で利用するフロントエンドサーバとバックエンドサーバの構築も行った。さらに筆者は、チームリーダとして、プロジェクトのタスク管理やバージョン管理を行うツールの導入・管理と、事務担当者に対する事務連絡などを行った。

結論としては、Tunnel 株式会社からのフィードバックにあるように、Tunnel 株式会社の要求を満たすツールを開発することはできた。しかし、このツールだけでは、まだ施策に結びつくまでの知見を与えることができない。これは、あくまで ARC はユーザの動向を把握することだけに特化したツールであり、RoomClip の運営に対して示唆を与えるようなツールになっていないためだと考えられる。さらに、試行錯誤を繰り返した開発であったため、ツールを製品として考えた場合、製品のテストをユーザテストしか行っておらず、十分な品質が確保できているとは言えない。また、ユーザインタフェースに関しても、なるべく Tunnel 株式会社からのユーザビリティの要求には答えたが、多くの人のユーザテストは行っていない。

今後の展望としては、品質とユーザビリティを高め、ARC をより使いやすく信頼性の高いツールにすると同時に、ユーザ動向の可視化だけではなく、別のアプローチから RoomClip のユーザ利用データを分析し、Tunnel 株式会社に対して RoomClip 運営の知見を示唆できるようなツールにしていくことが考えられる。今回は、各ユーザの行動だけに焦点を当て、ユーザ間のつながりについては考慮しなかった。岡本ら [10] によると、新規ユーザは SNS の中でも活発に活動していて友人が多いユーザとつながる可能性が高いため、新規ユーザが SNS に

定着するには、活発に活動しているユーザの行動とそのつながりについても分析する必要があると考えられる。

謝辞

本プロジェクトを進めるにあたり、Tunnel 株式会社代表取締役の高重正彦様、同社開発担当執行役員の平山知宏様には、RoomClip の利用データの提供や開発に関するご助言など、多大なご支援を頂きました。この場を借りて、深く御礼申し上げます。

指導教員の田中二郎教授には、本報告書の執筆と発表において多くのご指導を頂きました。

課題担当教員の岡瑞起准教授、橋本康弘助教には、プロジェクト遂行のご支援や技術的なアドバイスなど、日頃から熱心にご指導を頂きました。深く感謝申し上げます。

また、柿沢敦子様、塚本純子様、佐々木淳子様には、出張に関する事務手続きなどで大変お世話になりました。

本プロジェクトメンバーである河越嵩介氏、崔野氏、万シャンシャン氏とは、チームメンバーとして議論を交わし、とても充実した日々を過ごすことができました。

最後に、プロジェクト期間中のみならず、大学院生活すべてを支えて下さった友人と家族に心より感謝申し上げ、謝辞とさせていただきます。

参考文献

[1] 総務省. ICT の進化によるライフスタイル・ワークスタイルの変化. 平成 26 年度版情報通信白書, 第 1 部, 第 4 章, 第 1 節, 第 1 項, 2014.

[2] 鳥海不二夫, 山本仁志, 諏訪博彦, 岡田勇, 和泉潔, 橋本康弘. 大量 SNS サイトの比較分析. 人工知能学会論文誌, Vol.25, No.1, pp.78-89, 2010.

[3] 松尾豊, 安田雪. SNS における関係形成原理 — mixi のデータ分析 —. 人工知能学会論文誌, Vol.22, No.5, pp.531-541, 2007.

[4] 高木 昇. インターネット上のユーザレビューへの「賛同数」に見られるべき乗分布とその要因. 九州産業大学商経論叢, Vol.54, No.1, pp31-47, 2013.

[5] Petr Suchánek, Kateřina Slaninova, Robert Bucki. Business intelligence as the support of decision-making processes in e-commerce systems environment. MPRA Paper, No.27313, 2010.

[6] Dmitry Namiot, Manfred Sneps-Sneppe. On-Micro-services Architecture. International Journal of Open Information Technologies, Vol.25, No.9, pp.24-27, 2014.

[7] Roy Thomas Fielding. Architectural Styles andthe Design of Network-based Software Architectures. UNIVERSITY OF CALIFORNIA, IRVINE DISSERTATION submitted in partial satisfaction of the requirements for the degree of DOCTOR OF PHILOSOPHY in Information and Computer Science, 2000.

[8] Leonard Richardson, Sam Ruby. RESTful Web Services. 山本 陽平 (監修), 株式会社クイーブ (翻訳), オライリー・ジャパン, 2007.

[9] 井垣宏, 福安直樹, 佐伯幸郎, 松本真佑, 楠本真二. アジャイルソフトウェア開発教育のためのチケットシステムを用いたプロジェクト定量的評価手法の提案. 情報処理学会論文誌, Vol.56, No.2, pp.701-713, 2015.

[10] 岡本健志, 田中秀幸. 地域 SNS のユーザー同士のつながり方に着目したネットワーク分析. 日本社会情報学会学会誌, Vol.21, No.1, pp.45-55, 2009.

付 録A　Tunnel株式会社からのフィードバック文書全文

ARCフィードバックシート

はじめに

サービスURLにあるRoomClip分析ツール「ARC」を使って、フィードバックを追記していくシートです。何か新しい発見があり次第随時追記していきます。

サービスURL

http://arc-front.tk/

Tunnelフィードバック

Rankingに対するフィードバック

どのような操作をし、どう思ったか

＊単純にどういう行動している人がいっぱいいるのかを確認した
　　└今まで「いいね」何人、「クリップ」何人という単独でみていたが、それがベン図のように見えたため、「主にどういう行動をユーザーがしているのか」が見渡せた
　　└ただし相関がみれるわけではないので、それ以上の分析・インサイトはなかった

＊ある日の登録者を絞り込み、登録日から1日毎に期間をずらして変動率をおっかけた
　　└初日からアクティビティは減る一方であることがわかった
　　└あるタイミングでアクティビティが増えるという状況がもしあるならば、例えば「本当は撮影するユーザーでも、10個くらいいいねしたりコメントしたり様子を見たあとに投稿する」ようなことが言えるかもしれない、と考えた。

＊ある期間に登録したユーザーのアクティビティがどう減少していくのかをデイリー単位でみようとしたが、できなかった
　　└きっと減少の速度がアクティビティによって変わるはずだ、という想定があった

＊どの行為をやっているユーザーが最後までのこる（ゼロにならない）のかをみた
　　└スティッキーに使っているユーザーがどのアクティビティをやっているのかを確認したかったが、結構変動が激しく、捉えられなかった

＊ランキングの推移を確認しようとした
　　└ある時に登録した人のランキング変位がどのくらいのタイミングでおこるのかを見ようとしたが、煩雑でできなかった

総評

「なにもない状態で分析しろ」といわれたら、やはりフローについて見たがる傾向にあると自己認識した。ただし、もし「いいね促進キャンペーン」などを特定の期間ではっていた場

合、どの程度変動するのかを期間登録者ベースでみれると、効果測定としてはとてもよいと感じた。
結果、ランキング機能は「標本の絞り込み」と「期間」が非常に重要で、
つまり「標本を絞り込んだ理由」か「期間を絞り込んだ理由」が分析の要になると思った。
具体的には下記2つ。

1. ある期間にキャンペーンを張った場合、それが効果があったのか？の確認
2. ある特殊な経路で入ってきた登録者の場合、それがどのように変遷したのか？の確認

のようなことに効果的だと言えそうだ。
その場合これは「どういうユーザー特性があるのか」といったような、現状分析にはあまり向いておらず、施策の評価として意味をもつ、ということなので、なにも施策をうっていない状態での仮説立脚には至らなかった。

Flowに対するフィードバック

どのような操作をし、どう思ったか

＊ある時写真投稿をした人の、それより前のアクティブ履歴を眺めた
　　└母数が多いため「いいね」によってしまっているが、「なにもしない」から突如写真を撮る人もいるということもわかり、非常に興味深かった
　　└ただし「黄金パターン」のようなものはわからず。（そもそもないのかも）

＊クリップした人とそうでない人で、後の行動に変化がおきるかをみた
　　└流出のランキングをみたくなったが、一応手作業でみれた。が、クリップした人のほうが有意に撮影する、みたいな情報までは辿りつけなかった

＊ある時何もしなかった人（見る、とそれ以外で絞り込んだ）がその後どうしたのか、またはその前何をしていたのかを眺めた
　　└特に傾向がつかめなかった…

総評

ほとんど3層分析をして、常に中央または右端の層を固定して調査していた。
つまり、「ある行動をしたユーザーのその後の行動」を気にしたのではなく、
「ある行動をしたユーザーのその前の行動」を気にしていた。
「ある行動をさせたい」と思った時、どのツボを刺激すればしてもらえるのか、を求めて色々と出力をしていた。
逆に「ある行動をしなくなった」人のみ、「その後どんな行動をしたのか」をみた。
休眠ユーザーがどんなアクティビティフックで起き上がるのか、を調べたかった。
結果、本質的には「ユーザーがアクティブになるために必要なアクションを分析しようとした」という分析姿勢だったと思われた。
施策に結びつくほどの分析はできなかったが、逆に言うと「顕著な傾向がない」ということがわかった。

開発要望

- ランキングの日付境界ロジックが、登録者絞り込みと期間1,2でことなるを統一して欲しい。（2015-12-01 ~ 2015-12-02 で12月1日分なのだが、期間だと 2015-12-01 ~ 2015-12-01で12月1日分） → 対応しました。2015-12-01 ~ 2015-12-01で12月1日分になるように統一しました。（古谷）
- ランキングで更新を押した時に、ソート条件を同時に更新して欲しい。→ ソート条件は更新後も反映されているはずです。（古谷）
- フローで流入・流出をソートしたい。→ 対応しました。（崔）
- フローで流入・流出の出元・出先ベースでの割合がでていて欲しい。
- フローで「何もしなかった人」グループを選択したい → グルーピング時にVIEWのアイコンをチェックすると、VIEWしていないグループが選択できるようになります。そのグループが「何もしなかった人」グループです。灰色にして強調することを検討中です。（崔）

付 録B　ノードID一覧対応表

nodeid対応表（v2.1）

nodeid	VIEW	LIKE	CLIP	FOLLOW	COMMENT	POST	FAVTAG
1	1	0	0	0	0	0	0
2	1	1	0	0	0	0	0
3	1	0	1	0	0	0	0
4	1	1	1	0	0	0	0
5	1	0	0	1	0	0	0
6	1	1	0	1	0	0	0
7	1	0	1	1	0	0	0
8	1	1	1	1	0	0	0
9	1	0	0	0	1	0	0
10	1	1	0	0	1	0	0
11	1	0	1	0	1	0	0
12	1	1	1	0	1	0	0
13	1	0	0	1	1	0	0
14	1	1	0	1	1	0	0
15	1	0	1	1	1	0	0
16	1	1	1	1	1	0	0
17	1	0	0	0	0	1	0
18	1	1	0	0	0	1	0
19	1	0	1	0	0	1	0
20	1	1	1	0	0	1	0
21	1	0	0	1	0	1	0
22	1	1	0	1	0	1	0
23	1	0	1	1	0	1	0
24	1	1	1	1	0	1	0
25	1	0	0	0	1	1	0
26	1	1	0	0	1	1	0
27	1	0	1	0	1	1	0
28	1	1	1	0	1	1	0

nodeid对应表（v2.1）

29	1	0	0	1	1	1	0
30	1	1	0	1	1	1	0
31	1	0	1	1	1	1	0
32	1	1	1	1	1	1	0
33	1	0	0	0	0	0	1
34	1	1	0	0	0	0	1
35	1	0	1	0	0	0	1
36	1	1	1	0	0	0	1
37	1	0	0	1	0	0	1
38	1	1	0	1	0	0	1
39	1	0	1	1	0	0	1
40	1	1	1	1	0	0	1
41	1	0	0	0	1	0	1
42	1	1	0	0	1	0	1
43	1	0	1	0	1	0	1
44	1	1	1	0	1	0	1
45	1	0	0	1	1	0	1
46	1	1	0	1	1	0	1
47	1	0	1	1	1	0	1
48	1	1	1	1	1	0	1
49	1	0	0	0	0	1	1
50	1	1	0	0	0	1	1
51	1	0	1	0	0	1	1
52	1	1	1	0	0	1	1
53	1	0	0	1	0	1	1
54	1	1	0	1	0	1	1
55	1	0	1	1	0	1	1
56	1	1	1	1	0	1	1
57	1	0	0	0	1	1	1

nodeid対応表（v2.1）

58	1	1	0	0	1	1	1
59	1	0	1	0	1	1	1
60	1	1	1	0	1	1	1
61	1	0	0	1	1	1	1
62	1	1	0	1	1	1	1
63	1	0	1	1	1	1	1
64	1	1	1	1	1	1	1
65	0	0	0	0	0	0	0
66	新規登録ユーザ（flow/allのみ使用）						

付 録C　JavaScriptライブラリ比較表

JSフレームワークの比較

	機能の多さ	学習コストの低さ	ドキュメントの多さ	スケーラビリティ	保守性	備考
VanillaJS	○	○	○	×	×	DOM操作はFWを使うより断然速い
AngularJS (1.x)	○ フルスタック MVCモデル	× HTML側のやることが多い 覚えることがやや多い	○ 大規模フロントでは主流 ユーザが多い	○ MVCモデルでフレキシブル	× HTML側も修正する必要あり コードが複雑になりがち	
React.js	△ 仮想DOMやステートなどが特	△ 構文は単純だが、 独特なデータフローの理解が必	○ 大規模フロントでは主流 ユーザが多い	△ モジュールがしっかり 設計できていれば	△ ステートがあるので設計次第	パフォーマンスが良い
Vue.js	× "Just a View"	○ シンプルでわかりやすい 覚えることも多くない	△ まだまだ未成熟だが 注目はされている	△ モジュール化を自分でやる	△ シンプルなので見通しは良い	
Backbone.js	○ フルスタック MVCモデル	× 覚えることがやや多い	△ AngularやReactに比べる やや少ない	○ MVCモデルでフレキシブル モジュール化もしっかり	○ モジュール化が特長	
Ember.js	△ 他のFWにない機能がある	× 独特の思想の理解が必要	○ コミュニティが大きい 公式が良く出来ている	× 「意図的かつ構造的」に フレキシブルではない	× 設計が独特	"The Ember Way"という 独自の考え方
Knockout.js	× "Just a View"	○ シンプルでわかりやすい Vue.jsに近いかも	× 最近は落ち目らしい	× 中規模以上は苦手	△ しっかり設計すれば見通しは良い	
Aurelia.js	○ フルスタック MVCモデル	× 覚えることがやや多い？	× 2015/01に作られたばかり	○ 拡張性を考慮して作られている	△ Angularの短所を改善？	
参考						
http://gihyo.jp/dev/serial/01/emberjs/0001						
http://paiza.hatenablog						
http://www.buildinsider.net/web/popularjslib/2014						
http://dev.classmethod.jp/client-side/javascript/js-framework/						

付 録**D**　ユーザシナリオ一覧

古谷
1.
　ある日、平山さんがTunnel社に出社し、Google Analytics ToolでRoomClipのページビューを確認すると、先週よりもPV数が増えていることに気がついた。
　なぜか考えた時に、「RoomClipのコメントが炎上しているのか？」「雑誌がブログで取り上げられたのか？」「Twitterで大量にRTされているのか？」などといろいろと原因を考えたが、どれが決定的な原因かはわからない。
　そこで、筑波大学の学生が開発したARCを立ち上げ、ランキング画面を開いてみた。先週１週間と今週１週間の違いを見るため、スパンを7に設定し、先週の日付と今週の日付を入力し、ランキングを表示した。次に、何のアクションを起こした人が増えたのかをとりあえず確認するため、変動人数が多い順でソートし、それぞれのアクションごとでどれくらい人数が増えたのかを１つずつ確認していった。
　すると、COMMENTのアクションを行った人が特に増えていることに気がついたので、RoomClip内のコメント機能の中で何かしら炎上が起きている可能性があるということが推測された。
（そこで実際にどの写真が炎上しているのかを確認して、特定のタグがついた写真が炎上していることがわかったので、それに対して対処を行うことが出来た。）

2.
　ある月末、平山さんは、先月にRoomClipに追加した「ユーザが写真を投稿することを促す機能」の効果を測定するため、先月と今月で写真の投稿者数がどれくらい増えたかを確認することになった。
　そこで、ARCを立ち上げ、先月の1日と今月の1日を入力し、スパンを1ヶ月に設定した。さらに、グルーピング機能でPOSTにチェックを入れ、POSTしている人としていない人の2種類にユーザを分割し確認してみた。
　すると、POSTしている人もしていない人も増えてはいたが、POSTしている人のほうが変動率が高いことがわかり、写真を投稿しているユーザが増加傾向にあることが確認できた。

万
1、RoomClipのサービスが立ち上げてからN周年記念のため、Tunnel社が一ヶ月の期間のイベントを開催する。イベント期間中、利用者が投稿した写真数が１０枚以上、一枚あたりの写真が得られた「Like」は３０人以上のユーザーに対し、アンケートを配り、アンケートに記入した方は、Tunnel社からハガキや家具メーカーのクーポンをもらえる。このようなイベントに伴う、平山さんはRoomClipの写真の投稿数と「Like」を押した回数がはどれぐらい増えたかを確認するため、ARCを立ち上げ、イベントが開催された一ヶ月の期間を指定し、グルーピング機能でPOSTとLIKEにクリックを入れ、この一ヶ月間に、投稿数と「Like」数は大量に増えた、その中「Like」数より投稿数の変動率が大きいことがわかり、イベントの開催により、利用者のアクティビティーをあげられた。

２、同じくインテリアをシェアするウェブサービスである「HouseClip」は最近ますます人気になったと平山さんが気がづいた、「うちの利用者が奪われるか」「HouseClipのせいで、新規登録者が減るか」と平山さんは心配になった、そこで「ARC」を立ち上げ、最近RoomClipの運営状況をチェックする、FLOWで９月１日の前後３０日のスパンを指定し、９月１からの３０日間の新規登録者の人数が増えたが、変動範囲は先月より小さくなったことがわかった。「HouseClip」は「RoomClip」に影響をあたえられたと判定できる。

３、９月からRoomClipのコンメント機能にスタンプを貼りつる機能を追加した。利用者はコメントするときにRoomClipが提供するスタンプを入力できるようになった。平山さんは、この新しい機能の追加に伴う、利用者たちのコメント数が増えたか確認するため、ARCを立ち上げ、９月1日から９月３０日のスパンを指定し、さらに、グルーピング機能でCOMMENTにクリックを入れ、FLOW図からコメントした人が増えたことがわかり、スタンプ機能の追加により、利用者のコメント意欲を促せることを確認できた。

崔
背景
「クリップ機能は何なのかよくわからない」という書き込みを2chのRoomClip版で発見した。そこで運営側は、クリップ機能の利用率向上を狙って、「クリップを使ってみよう」とのバナーを掲載した。
「クリップ機能は何なのかよくわからない」という書き込みを2chのRoomClip串で発見した。そこで運営側は、クリップ機能の利用率向上を狙って、「クリップを使ってみよう」とのバナーを掲載した。
問題点
一ヶ月経った時点で、クリップ機能の使用状況を分かりやすいグラフでみたい
一ヶ月経った時点で、クリップ機能の使用状況を分かりやすいグラフでみたい
問題解決
1.ARCのflowページの開いて、バナー掲載前の一ヶ月とバナー掲載後の一ヶ月のフローをみる
2.「clip」でグルーピング
3.ポインターをブロックの上に移動することで、宣伝前後の、clip機能の絶対人数が分かる
4.前の一ヶ月で「clip」機能を使ったユーザ（8000人）は、後の一ヶ月もclip機能を使ったユーザは6000人（75%）であることが分かる（ブロックをクリックして、流出線の上にポインターを移動）
5.前の一ヶ月で「clip」機能を使わなかったユーザ(30000人)の中で、後一ヶ月でclipを使ったユーザは10000人であることが分かる（上と同様）

●河越
Tunnel社はある施策を行い、ある程度まとまった数の新規ユーザの獲得に成功した。今まで新規ユーザの獲得には難航していたので、今回獲得した新規ユーザには何としてもRoomClipを使い続けてもらいたい。そこで今回獲得した新規ユーザの内の何人が来週もRoomClipを使い続けてくれるかどうかを調べ、来週も使い続けてくれるユーザが少ないのであれば明日明後日にでも新たな施策を実施したい。この要求を解決するために機械学習システムを利用する。AWSで自動的に取得している今週のユーザのアクティビティのログを入力として与えることで、そのユーザが来週もRoomClipを使い続けるかどうかがわかる。今回獲得した新規ユーザのアクティビティのログを入力すると、その内の７割は来週は使わなくなるという結果が出力された。そこでTunnel社は新規ユーザにRoomClipを長く使い続けてもらうために新たな施策を打とうとミーティングを行うのだった。

付 録E　WebAPIのSQL一覧

Listing E.1: sql_ranking

```
SELECT
n_id AS nodeid,
COUNT(*) AS cnt
FROM
(
    SELECT
        u_id,
        n_id
    FROM
        (
            SELECT
                u_id,
                SIGN(SUM(u_view)) AS views,
                SIGN(SUM(u_like)) AS likes,
                SIGN(SUM(u_clip)) AS clips,
                SIGN(SUM(u_follow)) AS follows,
                SIGN(SUM(u_comment)) AS comments,
                SIGN(SUM(u_post)) AS posts,
                SIGN(SUM(u_favtag)) AS favtags
            FROM
                %(table)s
            WHERE
                u_date >= %(start)s
            AND u_date < %(end)s
            GROUP BY
                u_id
        ) r
        LEFT JOIN node2
        ON  r.views = node2.n_view
        AND r.likes = node2.n_like
        AND r.clips = node2.n_clip
        AND r.follows = node2.n_follow
        AND r.comments = node2.n_comment
        AND r.posts = node2.n_post
        AND r.favtags = node2.n_favtag
        WHERE n_id IS NOT NULL
) n
GROUP BY
nodeid
```

Listing E.2: sql_flow

```
SELECT
    COUNT(*) AS cnt,
    range1.n_id AS nodeid_range1,
    range2.n_id AS nodeid_range2
FROM
    (
```

```
        SELECT
            u_id,
            n_id
        FROM
            (
                SELECT
                    u_id,
                    SIGN(SUM(u_view)) AS views,
                    SIGN(SUM(u_like)) AS likes,
                    SIGN(SUM(u_clip)) AS clips,
                    SIGN(SUM(u_follow)) AS follows,
                    SIGN(SUM(u_comment)) AS comments,
                    SIGN(SUM(u_post)) AS posts,
                    SIGN(SUM(u_favtag)) AS favtags
                FROM
                    %(table)s
                WHERE
                    u_date >= %(start1)s
                AND u_date < %(end1)s
                GROUP BY
                    u_id
            ) r
            LEFT JOIN node2
            ON  r.views = node2.n_view
            AND r.likes = node2.n_like
            AND r.clips = node2.n_clip
            AND r.follows = node2.n_follow
            AND r.comments = node2.n_comment
            AND r.posts = node2.n_post
            AND r.favtags = node2.n_favtag
            WHERE n_id IS NOT NULL
    ) range1,
    (
        SELECT
            u_id,
            n_id
        FROM
            (
                SELECT
                    u_id,
                    SIGN(SUM(u_view)) AS views,
                    SIGN(SUM(u_like)) AS likes,
                    SIGN(SUM(u_clip)) AS clips,
                    SIGN(SUM(u_follow)) AS follows,
                    SIGN(SUM(u_comment)) AS comments,
                    SIGN(SUM(u_post)) AS posts,
                    SIGN(SUM(u_favtag)) AS favtags
                FROM
                    %(table)s
                WHERE
                    u_date >= %(start2)s
                AND u_date < %(end2)s
                GROUP BY
                    u_id
            ) r
            LEFT JOIN node2
            ON  r.views = node2.n_view
            AND r.likes = node2.n_like
            AND r.clips = node2.n_clip
            AND r.follows = node2.n_follow
            AND r.comments = node2.n_comment
            AND r.posts = node2.n_post
```

```
            WHERE n_id IS NOT NULL
    ) range2
WHERE
    range1.u_id = range2.u_id
GROUP BY
    range1.n_id,
    range2.n_id
```

Listing E.3: sql_flow_all

```
SELECT
    COUNT(*) AS cnt,
    nr1_nid,
    nr2_nid
FROM
    (
        SELECT
            nr2.u_id,
            COALESCE(nr1.n_id, 66) AS nr1_nid,
            nr2.n_id AS nr2_nid
        FROM
            (
                SELECT
                    u_id,
                    COALESCE(n_id, 65) AS n_id
                FROM
                    (
                        SELECT
                            ui1.id AS u_id,
                            SIGN(SUM(ua1.u_view)) AS views,
                            SIGN(SUM(ua1.u_like)) AS likes,
                            SIGN(SUM(ua1.u_clip)) AS clips,
                            SIGN(SUM(ua1.u_follow)) AS follows,
                            SIGN(SUM(ua1.u_comment)) AS comments,
                            SIGN(SUM(ua1.u_post)) AS posts,
                            SIGN(SUM(ua1.u_favtag)) AS favtags
                        FROM
                            (
                                SELECT
                                    *
                                FROM
                                    user_info
                                WHERE
                                    TO_CHAR(created, 'YYYY-MM-DD') < %(created2)s
                            ) ui1
                            LEFT JOIN(
                                    SELECT
                                        *
                                    FROM
                                        %(table)s
                                    WHERE
                                        u_date >= %(start2)s
                                    AND u_date < %(end2)s
                                ) ua1
                            ON  ui1.id = ua1.u_id
                        GROUP BY
                            ui1.id
                    ) r2
                    LEFT JOIN node2
                    ON  r2.views = node2.n_view
                    AND r2.likes = node2.n_like
                    AND r2.clips = node2.n_clip
```

```
                    AND r2.follows = node2.n_follow
                    AND r2.comments = node2.n_comment
                    AND r2.posts = node2.n_post
                    AND r2.favtags = node2.n_favtag
            ) nr2
            LEFT JOIN(
                    SELECT
                        u_id,
                        COALESCE(n_id, 65) AS n_id
                    FROM
                        (
                            SELECT
                                ui2.id AS u_id,
                                SIGN(SUM(ua2.u_view)) AS views,
                                SIGN(SUM(ua2.u_like)) AS likes,
                                SIGN(SUM(ua2.u_clip)) AS clips,
                                SIGN(SUM(ua2.u_follow)) AS follows,
                                SIGN(SUM(ua2.u_comment)) AS comments,
                                SIGN(SUM(ua2.u_post)) AS posts,
                                SIGN(SUM(ua2.u_favtag)) AS favtags
                            FROM
                                (
                                    SELECT
                                        *
                                    FROM
                                        user_info
                                    WHERE
                                        TO_CHAR(created, 'YYYY-MM-DD') < %(created1)s
                                ) ui2
                                LEFT JOIN(
                                        SELECT
                                            *
                                        FROM
                                            user_activity2
                                        WHERE
                                            u_date >= %(start1)s
                                        AND u_date < %(end1)s
                                    ) ua2
                                ON  ui2.id = ua2.u_id
                            GROUP BY
                                ui2.id
                        ) r1
                        LEFT JOIN node2
                        ON  r1.views = node2.n_view
                        AND r1.likes = node2.n_like
                        AND r1.clips = node2.n_clip
                        AND r1.follows = node2.n_follow
                        AND r1.comments = node2.n_comment
                        AND r1.posts = node2.n_post
                        AND r1.favtags = node2.n_favtag
                ) nr1
            ON  nr2.u_id = nr1.u_id
    ) nr
GROUP BY
    nr1_nid,
    nr2_nid
```

Listing E.4: sql_flow_registration

```
SELECT
    COUNT(*) AS cnt,
    range1.n_id AS nodeid_range1,
```

```
    range2.n_id AS nodeid_range2
FROM
    (
        SELECT
            u_id,
            n_id
        FROM
            (
                SELECT
                    u_id,
                    SIGN(SUM(u_view)) AS views,
                    SIGN(SUM(u_like)) AS likes,
                    SIGN(SUM(u_clip)) AS clips,
                    SIGN(SUM(u_follow)) AS follows,
                    SIGN(SUM(u_comment)) AS comments,
                    SIGN(SUM(u_post)) AS posts,
                    SIGN(SUM(u_favtag)) AS favtags
                FROM
                    %(table)s
                    LEFT JOIN user_info
                    ON %(table)s.u_id = user_info.id
                WHERE
                    u_date >= %(start1)s
                AND u_date < %(start2)s
                AND TO_CHAR(created, 'YYYY-MM-DD') BETWEEN %(startreg)s AND %(endreg)s
                GROUP BY
                    u_id
            ) r
            LEFT JOIN node2
            ON  r.views = node2.n_view
            AND r.likes = node2.n_like
            AND r.clips = node2.n_clip
            AND r.follows = node2.n_follow
            AND r.comments = node2.n_comment
            AND r.posts = node2.n_post
            AND r.favtags = node2.n_favtag
            WHERE n_id IS NOT NULL
    ) range1,
    (
        SELECT
            u_id,
            n_id
        FROM
            (
                SELECT
                    u_id,
                    SIGN(SUM(u_view)) AS views,
                    SIGN(SUM(u_like)) AS likes,
                    SIGN(SUM(u_clip)) AS clips,
                    SIGN(SUM(u_follow)) AS follows,
                    SIGN(SUM(u_comment)) AS comments,
                    SIGN(SUM(u_post)) AS posts,
                    SIGN(SUM(u_favtag)) AS favtags
                FROM
                    %(table)s
                    LEFT JOIN user_info
                    ON %(table)s.u_id = user_info.id
                WHERE
                    u_date >= %(start2)s
                AND u_date < %(end2)s
                AND TO_CHAR(created, 'YYYY-MM-DD') BETWEEN %(startreg)s AND %(endreg)s
                GROUP BY
```

```
                    u_id
            ) r
            LEFT JOIN node2
            ON  r.views = node2.n_view
            AND r.likes = node2.n_like
            AND r.clips = node2.n_clip
            AND r.follows = node2.n_follow
            AND r.comments = node2.n_comment
            AND r.posts = node2.n_post
            WHERE n_id IS NOT NULL
    ) range2
WHERE
    range1.u_id = range2.u_id
GROUP BY
    range1.n_id,
    range2.n_id
```

Listing E.5: sql_transition_actions

```
SELECT
    TO_CHAR(u_date, 'YYYY-MM-DD') AS date,
    COUNT(CASE WHEN u_view > 0 THEN 1 ELSE NULL END) AS views,
    COUNT(CASE WHEN u_like > 0 THEN 1 ELSE NULL END) AS likes,
    COUNT(CASE WHEN u_clip > 0 THEN 1 ELSE NULL END) AS clips,
    COUNT(CASE WHEN u_follow > 0 THEN 1 ELSE NULL END) AS follows,
    COUNT(CASE WHEN u_comment > 0 THEN 1 ELSE NULL END) AS comments,
    COUNT(CASE WHEN u_post > 0 THEN 1 ELSE NULL END) AS posts,
    COUNT(CASE WHEN u_favtag > 0 THEN 1 ELSE NULL END) AS favtags
FROM
    (
        SELECT
            *
        FROM
            (
                (
                    SELECT
                        *
                    FROM
                        %(table)s
                    WHERE
                        u_date BETWEEN %(start)s AND %(end)s
                    GROUP BY
                        u_date,
                        u_id
                ) ua
                LEFT JOIN
                    (
                        SELECT
                            id,
                            TO_CHAR(created, 'YYYY-MM-DD') AS created
                        FROM
                            user_info
                    ) ui
                ON  ua.u_id = ui.id
            )
        WHERE
            created BETWEEN %(startreg)s AND %(endreg)s
    ) uia
GROUP BY
    u_date
```

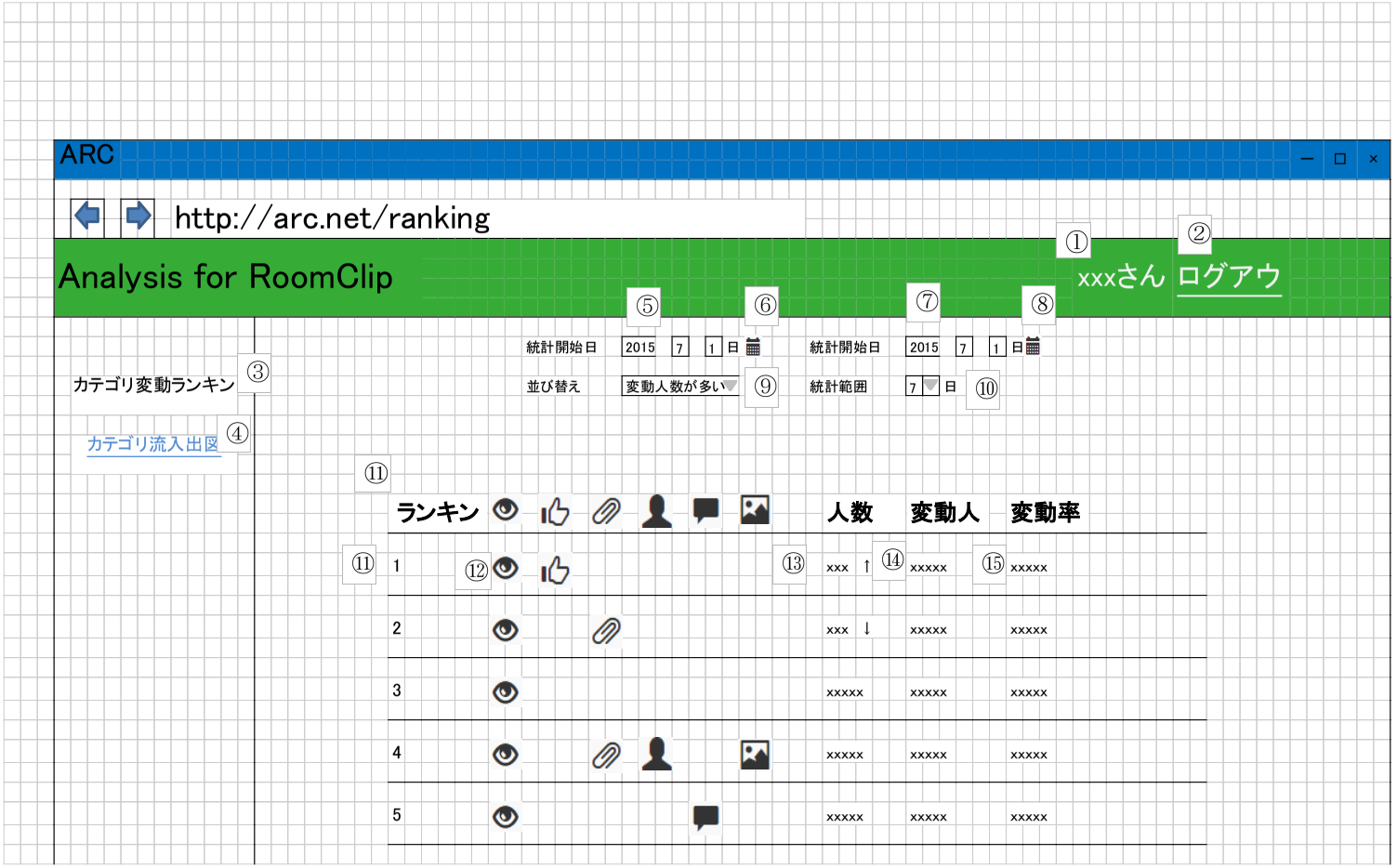
Listing E.6: sql_transition_registrations

```
SELECT
    TO_CHAR(created, 'YYYY-MM-DD') AS date,
    COUNT(*) AS registrations
FROM
    user_info
WHERE
    TO_CHAR(created, 'YYYY-MM-DD') BETWEEN %(start)s AND %(end)s
GROUP BY
    TO_CHAR(created, 'YYYY-MM-DD')
ORDER BY
    TO_CHAR(created, 'YYYY-MM-DD') ASC
```

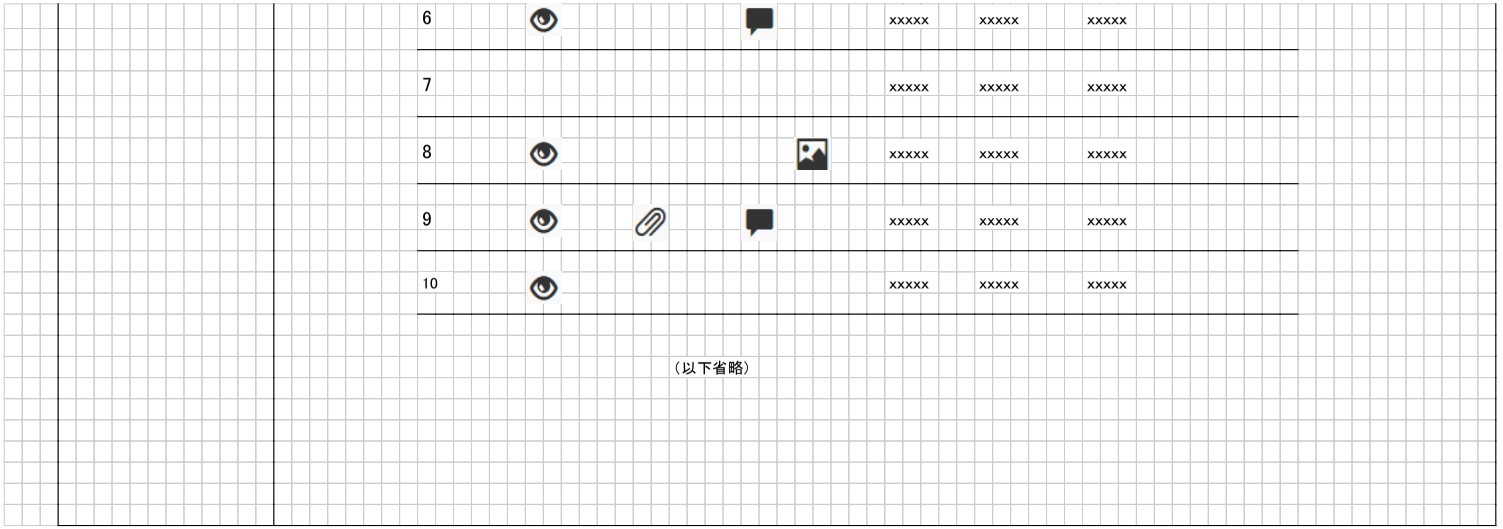
Listing E.7: sql_transition_sum

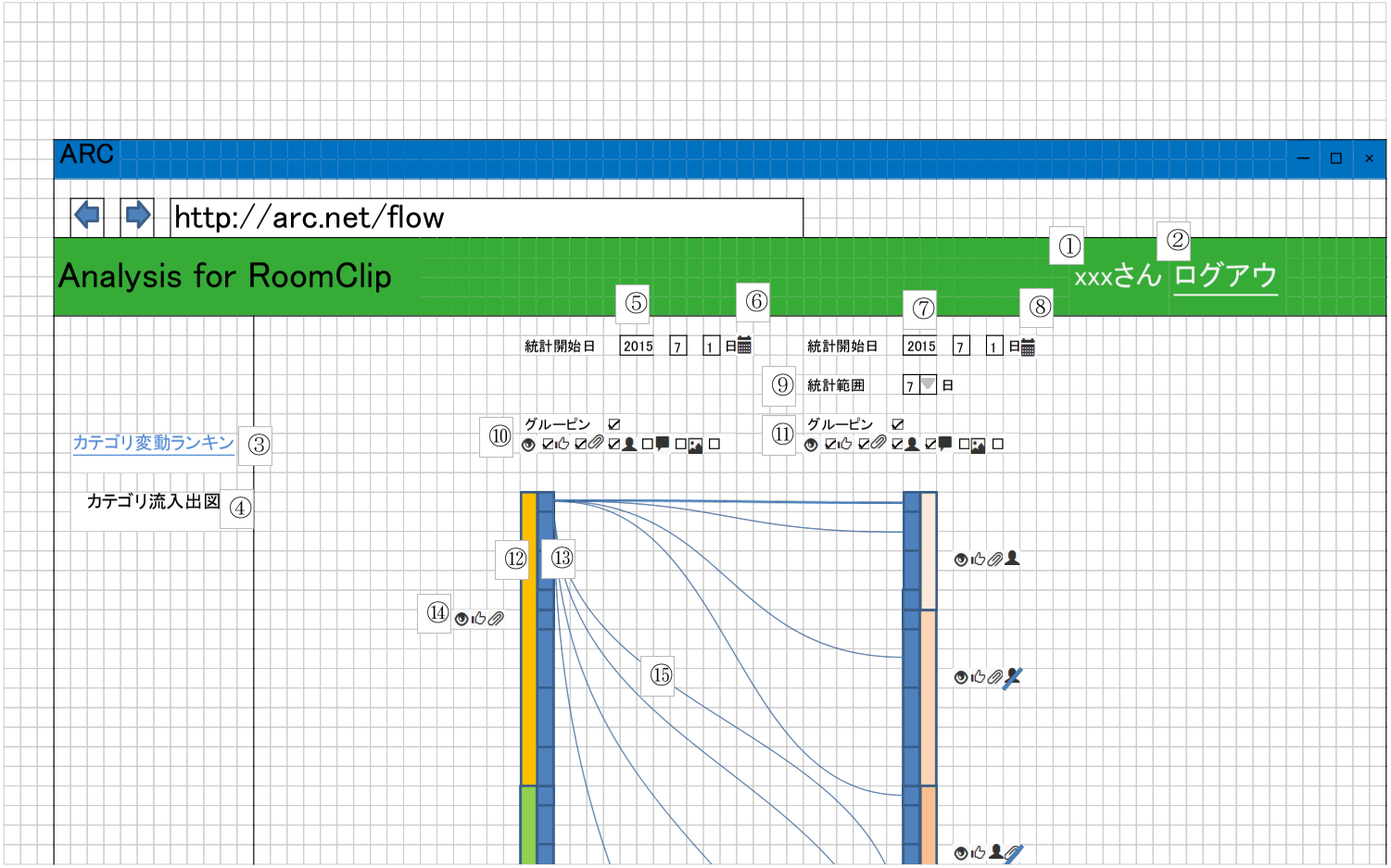
```
SELECT
    TO_CHAR(u_date, 'YYYY-MM-DD') AS date,
    SUM(u_view) AS views,
    SUM(u_like) AS likes,
    SUM(u_clip) AS clips,
    SUM(u_follow) AS follows,
    SUM(u_comment) AS comments,
    SUM(u_post) AS posts,
    SUM(u_favtag) AS favtags
FROM
    %(table)s
GROUP BY
    TO_CHAR(u_date, 'YYYY-MM-DD')
ORDER BY
    TO_CHAR(u_date, 'YYYY-MM-DD') ASC
```

付 録F　画面設計書

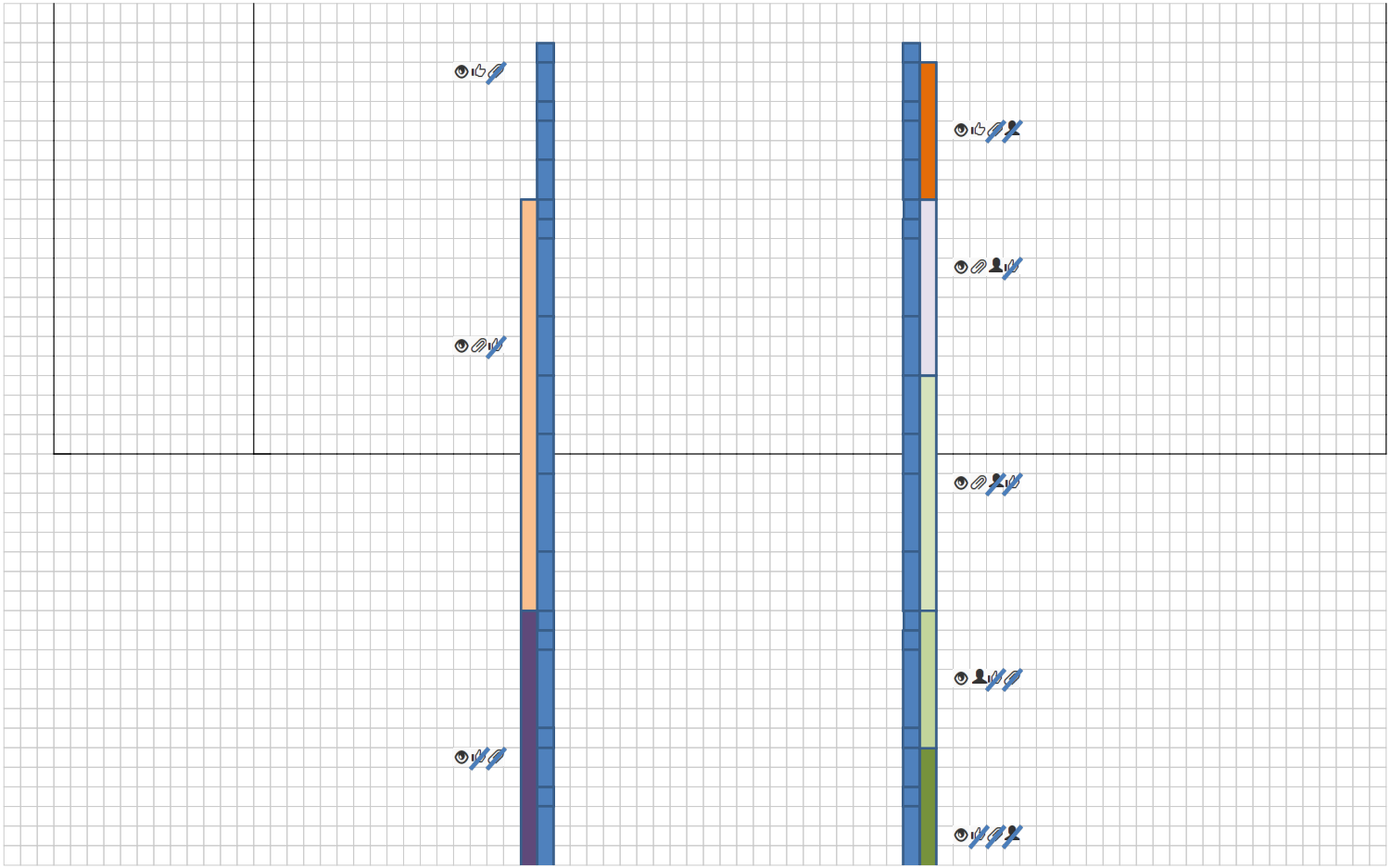

ARC
http://arc.net/ranking
Analysis for RoomClip
①
xxxさん
②
ログアウト
⑤
⑥
⑦
⑧
統計開始日
2015
7
1
日
統計開始日
2015
7
1
日
カテゴリ変動ランキン
③
並び替え
変動人数が多い
⑨
統計範囲
7
日
⑩
カテゴリ流入出図
④
⑪
ランキン
人数
変動人
変動率
⑪
1
⑫
⑬
xxx
↑
⑭
xxxxx
⑮
xxxxx
2
xxx
↓
xxxxx
xxxxx
3
xxxxx
xxxxx
xxxxx
4
xxxxx
xxxxx
xxxxx
5
xxxxx
xxxxx
xxxxx

画面設計図.xlsx


6
xxxxx
xxxxx
xxxxx
7
xxxxx
xxxxx
xxxxx
8
xxxxx
xxxxx
xxxxx
9
xxxxx
xxxxx
xxxxx
10
xxxxx
xxxxx
xxxxx
（以下省略）


ARC
http://arc.net/flow
Analysis for RoomClip
①
xxxさん
②
ログアウト
⑤
⑥
⑦
⑧
統計開始日
2015
7
1
日
統計開始日
2015
7
1
日
⑨
統計範囲
7
日
⑩
グルーピン
⑪
グルーピン
カテゴリ変動ランキン
③
カテゴリ流入出図
④
⑫
⑬
⑭
⑮

画面設計図.xlsx



画面設計図.xlsx

画面設計図.xlsx

ID	タイプ	説明	備考				
1	文字列	ユーザ名					
2	リンク	ユーザログアウト用リンク					
3	リンク	画面001に移動する					
4	文字列	現在の画面の名前					
5	テキストボックス	統計開始日1を入力する	統計開始日1と統計開始日2において、開始日1が開始日2の後ろになる組み合わせは不可				
6	ボタン	カレンダーから統計開始日1を選択する					
7	テキストボックス	統計開始日2を入力する					
8	ボタン	カレンダーから統計開始日2を選択する					
9	テキストボックス＆プルダウンリ	統計SPAN設定用	直接入力（1以上の整数）も可、プルダウンリストから選択（1、7，14，30）も可				
10	チェックボックス組	統計範囲1のグルーピング					
11	チェックボックス組	統計範囲2のグルーピング					
12							
13							

画面設計図.xlsx

[1] 作成者:
上がる（↑）下げる（↓）が分かるように

[2] 作成者:
序列があればいいかも

[3] 作成者:
とりあえず３３＊３３全部載せる

77

付 録G　テーブル仕様書

テーブル仕様書(v3.0)

テーブル情報

	テーブル型	ファクトテーブル	**作成者**	河越 嵩介
	論理テーブル名	ユーザアクティビ	**作成日**	2015/08/06
	物理テーブル名	user_activity	**更新日**	2015/10/08
	スキーマ	public		
	【備考】			
	その日（date）に行われた各ユーザの能動的な行動の合計数を格納す			

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only one	備考
1	日付	u_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	u_id	INTEGER	Yes	2		
3	view数	u_view	SMALLINT	Yes			これが1以上、つまり一回でも画面を見たユーザのアクティビティだけがこのテーブルに格納されている
4	like数	u_like	SMALLINT	Yes			
5	clip数	u_clip	SMALLINT	Yes			
6	follow数	u_follow	SMALLINT	Yes			
7	comment数	u_comment	SMALLINT	Yes			
8	post数	u_post	SMALLINT	Yes			
9	favtag数	u_favtag	SMALLINT	Yes			2.0から追加

外部カラム情報

No	外部カラム名	参照先テーブル名	参照先カラムリスト	期間	備考
3	u_view	daily_count.view	v_count	2014-10-08 ~ 2015-08-01	viewはaccessを元に作成
4	u_like	daily_count.like	l_count	2012-04-12 ~ 2015-05-21	likeはphoto_moreを元に作成
5	u_clip	daily_count.clip	cl_count	2013-12-05 ~ 2015-05-21	clipはphoto_clipsを元に作成
6	u_follow	daily_count.follow	f_count	2012-04-09 ~ 2015-05-21	followはuser_followを元に作成
7	u_comment	daily_count.comme	cm_count	2012-04-12 ~ 2015-05-21	commentはphoto_commentを元に作成
8	u_post	daily_count.post	ps_count	2012-04-12 ~ 2015-05-21	postはphotosを元に作成
9	u_favtag	daily_count.fav_tag	ft_count	2012-04-09 ~ 2015-05-21	favtagはfav_tag（request）を元に作成

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ファクトテーブル	**作成者**	河越 嵩介	
論理テーブル名	ビュー	**作成日**	2015/08/11	
物理テーブル名	view	**更新日**	2015/08/19	
スキーマ	daily_count			
【備考】				
その日（date）の各ユーザのviewの回数。raw_data.accessから抽				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only o	備考
1	日付	v_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	v_u_id	INTEGER	Yes	2		
3	view数	v_count	INTEGER	Yes			

外部カラム情報

No	外部カラム名	参照先テーブル名	参照先カラムリスト	期間	備考
1	v_date	raw_data.access	a_date	2014-10-07 ~ 2015-08-01	
2	v_u_id	raw_data.access	a_user_id	2014-10-07 ~ 2015-08-01	
3	v_count	raw_data.access	count(*)	2014-10-07 ~ 2015-08-01	

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ファクトテーブル	**作成者**	河越 嵩介	
論理テーブル名	ライク	**作成日**	2015/08/18	
物理テーブル名	like	**更新日**	2015/08/18	
スキーマ	daily_count			
【備考】				
その日（date）の各ユーザのlikeの回数。rwa_data.likeから抽出				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only o	備考
1	日付	l_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	l_u_id	INTEGER	Yes	2		
3	like数	l_count	INTEGER	Yes			

外部カラム情報

No	外部カラム名	参照先テーブル名	参照先カラムリスト	期間	備考
1	l_date	raw_data.like	l_date	2012-04-12 ~ 2015-05-21	
2	l_u_id	raw_data.like	l_u_id	2012-04-12 ~ 2015-05-21	
3	l_count	raw_data.like	count(*)	2012-04-12 ~ 2015-05-21	

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ファクトテーブル	**作成者**	河越 嵩介	
論理テーブル名	クリップ	**作成日**	2015/08/11	
物理テーブル名	clip	**更新日**	2015/08/18	
スキーマ	daily_count			
【備考】				
その日（date）の各ユーザのclipの回数。rwa_data.clipから抽出				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only o	備考
1	日付	cl_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	cl_u_id	INTEGER	Yes	2		
3	clip数	cl_count	INTEGER	Yes			

外部カラム情報

No	外部カラム名	参照先テーブル名	参照先カラムリスト	期間	備考
1	cl_date	raw_data.clip	cl_date	2013-12-05 ~ 2015-05-21	
2	cl_u_id	raw_data.clip	cl_u_id	2013-12-05 ~ 2015-05-21	
3	cl_count	raw_data.clip	count(*)	2013-12-05 ~ 2015-05-21	

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ファクトテーブル	**作成者**	河越 嵩介	
論理テーブル名	フォロー	**作成日**	2015/08/18	
物理テーブル名	follow	**更新日**	2015/08/18	
スキーマ	daily_count			
【備考】				
その日（date）の各ユーザのfollowの回数。rwa_data.followから抽				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only o	備考
1	日付	f_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	f_u_id	INTEGER	Yes	2		
3	follow数	f_count	INTEGER	Yes			

外部カラム情報

No	外部カラム名	参照先テーブル名	参照先カラムリスト	期間	備考
1	f_date	raw_data.follow	f_date	2012-04-09 ~ 2015-05-21	
2	f_u_id	raw_data.follow	following_id	2012-04-09 ~ 2015-05-21	
3	f_count	raw_data.follow	count(*)	2012-04-09 ~ 2015-05-21	

テーブル仕様書(v3.0)

テーブル情報

	テーブル型	ファクトテーブル	**作成者**	河越 嵩介	
	論理テーブル名	コメント	**作成日**	2015/08/11	
	物理テーブル名	comment	**更新日**	2015/08/18	
	スキーマ	daily_count			
	【備考】				
	その日（date）の各ユーザのcommentの回数。rwa_data.commentか				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only o	備考
1	日付	cm_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	cm_u_id	INTEGER	Yes	2		
3	comment数	cm_count	INTEGER	Yes			

外部カラム情報

No	外部カラム名	参照先テーブル名	参照先カラムリスト	期間	備考
1	cm_date	raw_data.comment	cm_date	2012-04-12 ~ 2015-05-21	
2	cm_u_id	raw_data.comment	cm_u_id	2012-04-12 ~ 2015-05-21	
3	cm_count	raw_data.comment	count(*)	2012-04-12 ~ 2015-05-21	

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ファクトテーブル	**作成者**	河越 嵩介	
論理テーブル名	ポスト	**作成日**	2015/08/19	
物理テーブル名	post	**更新日**	2015/09/29	
スキーマ	daily_count			
【備考】				
その日（date）の各ユーザのpostの回数。raw_data.postから抽出				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only o	備考
1	日付	p_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	p_u_id	INTEGER	Yes	2		
3	post数	p_count	INTEGER	Yes			

外部カラム情報

No	外部カラム名	参照先テーブル名	参照先カラムリスト	期間	備考
1	p_date	raw_data.post	p_date	2012-04-12 ~ 2015-05-21	
2	p_u_id	raw_data.post	p_user_id	2012-04-12 ~ 2015-05-21	
3	p_count	raw_data.post	count(*)	2012-04-12 ~ 2015-05-21	

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ファクトテーブ	**作成者**	河越 嵩介	
論理テーブル名	ファボタグ	**作成日**	2015/10/08	
物理テーブル名	fav_tag	**更新日**	2015/10/08	
スキーマ	daily_count			
【備考】				
その日（date）の各ユーザのviewの回数。raw_data.fav_tagから抽出				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only one)	備考
1	日付	ft_date	DATE	Yes	1	Yes	xxxx-yy-zz
2	ユーザID	ft_u_id	INTEGER	Yes	2		
3	view数	ft_count	INTEGER	Yes			

外部カラム情報

No	外部カラム名	参照先テーブル	参照先カラムリスト	期間	備考
1	ft_date	raw_data.fav_tag	ft_date	2012-04-09 ~ 2015-05-21	
2	ft_u_id	raw_data.fav_tag	ft_user_id	2012-04-09 ~ 2015-05-21	
3	ft_count	raw_data.fav_tag	count(*)	2012-04-09 ~ 2015-05-21	

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ディメンションテー	**作成者**	河越 嵩介		
論理テーブル名	ノード	**作成日**	2015/08/06		
物理テーブル名	node	**更新日**	2015/11/04		
スキーマ	public				
【備考】					
nodeidの対応表					

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only o	備考
1	nodeid	n_id	SMALLINT	Yes	1		
2	view	n_view	SMALLINT	Yes			
3	like	n_like	SMALLINT	Yes			
4	clip	n_clip	SMALLINT	Yes			
5	follow	n_follow	SMALLINT	Yes			
6	comment	n_comment	SMALLINT	Yes			
7	post	n_post	SMALLINT	Yes			
8	favtag	n_favtag	SMALLINT	Yes			Web版にはない

テーブル仕様書(v3.0)

テーブル情報

テーブル型	ファクトテーブル	**作成者**	河越 嵩介	
論理テーブル名	トレーニングデー	**作成日**	2015/10/23	
物理テーブル名	td_ua	**更新日**	2015/12/07	
スキーマ	public			
【備考】				
機械学習のための学習データ。user_activityにcount_daysが追加されている				

カラム情報

No	論理名	物理名	データ型	Not Null	sortkey	distkey(Only one)	備考
1	view数	view	SMALLINT	Yes			これが1以上、つまり一回でも画面を見たユーザのアクティビティだけがこのテーブルに格納されている
2	like数	likee	SMALLINT	Yes			
3	clip数	clip	SMALLINT	Yes			
4	follow数	follow	SMALLINT	Yes			
5	comment数	comment	SMALLINT	Yes			
6	post数	post	SMALLINT	Yes			
7	favtag数	favtag	SMALLINT	Yes			
8	7日の間にアクセスした日数	count_days	SMALLINT	Yes			

付 録H　API仕様書

API仕様（v3.3）

作成者：古谷翔太
更新日：2015-12-21（Dashboardの設計を変更）

GET ~/api/v3/ranking/{start1}/{start2}/{span}

今日から2週間前〜1週間前と1週間前から今日までを集計したランキングを返す

パラメタ名	説明
start1	起点となる日付1 (YYYY-MM-DD)
start2	起点となる日付2 (YYYY-MM-DD)
span	集計期間（起点となる日付を含める）

GET ~/api/v3/ranking/{start1}/{start2}/{span}/registration/{startreg}/{endreg}

指定した期間に新規登録したユーザに関して集計したランキングを返す

パラメタ名	説明
startreg	新規登録期間開始日
endreg	新規登録期間終了日
start1	起点となる日付1
start2	起点となる日付2
span	集計期間（起点となる日付を含める）

GET ~/api/v3/flow/{start1}/{start2}/{span}

集計したノード間流出入を返す

パラメタ名	説明
start1	起点となる日付1
start2	起点となる日付2
span	集計期間（起点となる日付を含める）

GET
~/api/v3/flow/{start1}/{start2}/{span}/registration/{startreg}/{endreg}

指定した期間に新規登録したユーザに関して集計したノード間流出入を返す

パラメタ名	説明
startreg	新規登録期間開始日
endreg	新規登録期間終了日
start1	起点となる日付1
start2	起点となる日付2
span	集計期間（起点となる日付を含める）

GET ~/api/v3/flow/{start1}/{start2}/{span}/all

集計したノード間流出入を返す＜NodeID65(noview), 66(new) 含む＞

パラメタ名	説明
start1	起点となる日付1
start2	起点となる日付2
span	集計期間（起点となる日付を含める）

GET ~/api/v3/transition/actions

全ユーザを対象とし、全期間の日付ごとに全ユーザ数と各アクションを行った人数を返す

GET ~/api/v3/transition/actions/{startreg}/{endreg}

startregからendregの間に登録したユーザを対象とし、全期間の日付ごとに全ユーザ数と各アクションを行った人数を返す

パラメタ名	説明
startreg	ユーザの登録範囲開始日
endreg	ユーザの登録範囲終了日

GET
~/api/v3/transition/actions/{startreg}/{endreg}/{start}/{end}

startregからendregの間に登録したユーザを対象とし、startからendまでの期間の日付ごとにユーザ数と各アクションを行った人数を返す

パラメタ名	説明
startreg	ユーザの登録範囲開始日
endreg	ユーザの登録範囲終了日
start	集計する範囲の開始日
end	集計する範囲の終了日

GET ~/api/v3/transition/registrations

全期間の日付ごとの登録者の推移を返す

GET ~/api/v3/transition/registrations/{start}/{end}

startからendまでの日付ごとの登録者の推移を返す

パラメタ名	説明
start	範囲の開始日
end	範囲の終了日

==

◆rankingのJSONフォーマット

```
{
	start1：2015-07-20, // 起点となる日付1
	start2：2015-07-27, // 起点となる日付2
	span：7, // start1,2からの集計期間(start1,2も含める)
	ranking[65]： [ {
		nodeid：4, // NodeID（下記参照）
		actions：[“VIEW”, “LIKE”, “POST”], // Actions（下記参照）
		range1：2000, // start1に関する集計の結果
		range2：2200 // start2に関する集計の結果
	} ]
}
```

◆flowのJSONフォーマット

```
{
	start1：2015-07-20, // 起点となる日付
	start2：2015-07-27, // 起点となる日付2
	span1：7, // start1,2からの集計期間(start1,2も含める)
	nodes[65]： [ {
```

```
            nodeid：4, // NodeID（下記参照）
            actions：[“VIEW”, “LIKE”, “POST”], // Actions（下記参照）
            range1：2000, // start1に関する集計の結果
            range2：2200 // start2に関する集計の結果
            flowout[65]：{ // 各ノードに対して出ていく数に関する情報
                  nodeid：2, // NodeID
                  actions：[“VIEW”, “COMMENT”], //Actions
                  num：100 // 出ていく数
            }]
}
```

◆flow/allのJSONフォーマット

```
{
      start1：2015-07-20, // 起点となる日付
      start2：2015-07-27, // 起点となる日付2
      span1：7, // start1,2からの集計期間(start1,2も含める)
      nodes[66]： [ {
            nodeid：4, // NodeID（下記参照）
            actions：[“VIEW”, “LIKE”, “POST”], // Actions（下記参照）
            range1：2000, // start1に関する集計の結果
            range2：2200 // start2に関する集計の結果
            flowout[66]：{ // 各ノードに対して出ていく数に関する情報
                  nodeid：2, // NodeID
                  actions：[“VIEW”, “COMMENT”], //Actions
                  num：100 // 出ていく数
            }]
}
```

◆transition/actionsのJSONフォーマット

```
{
      start: 2014-01-01, // 開始日
      end: 2015-01-01, // 終了日
      startreg: 2014-12-01 // 登録開始日
      endreg: 2014-12-01 // 登録終了日
      data：[{
            date: “2015-04-01”, // 日付
            views: “50” // 以下、この日に各アクションを行ったユーザ数
            likes: “50”
            clips: “50”
            follows: “50”
            comments: “50”
            posts: “50”
            favtags: “50”
      }]
}
```

◆transition/registrationのJSONフォーマット

```
{
	start: 2014-01-01, // 開始日
	end: 2015-01-01, // 終了日
	data : [{
		date: “2015-04-01”, // 日付
		registrations: “100” // 登録人数
	}]
}
```

==

Actions の定義

VIEW	写真詳細またはタイムラインを見る
LIKE	「いいね！」を押下
CLIP	写真をクリップ
FOLLOW	他ユーザをフォロー
COMMENT	写真にコメント
POST	写真を投稿
FAVTAG	タグをお気に入りに追加（モバイルのみ）

※ 空集合は**NOVIEW**（写真詳細またはタイムラインが見られていない）とする
※ flow/allのみ，**NEW**（新規登録）ノードが存在する

==

NodeIDとの対応関係

以下を参照
https://docs.google.com/spreadsheets/d/1T3nR2XL1Y5iunix2WHj0OuXpO2sdeh3Tchk2FWBPP4w/edit?usp=sharing

付 録I　サーバ設定書

フロントサーバ設定書
設定者：古谷

参考
http://blog.genies.jp/2011/08/amazon-ec2-amazon-linux.html
http://itpro.nikkeibp.co.jp/article/COLUMN/20080219/294153/
http://www.maruko2.com/mw/%E4%B8%80%E8%88%AC%E3%83%A6%E3%83%BC%E3%82%B6%E3%83%BC%E3%82%92_sudo_%E3%81%A7%E3%81%8D%E3%82%8B%E3%82%88%E3%81%86%E3%81%AB%E3%81%99%E3%82%8B
http://dev.classmethod.jp/etc/amazon_linux_nginx_basic_auth/

サーバ構築
Amazon EC2 で無料枠でインスタンスを作った
　　********@gmail.com で新しくアカウントを作った

OS は Amazon Linux を選択、ほかの設定はいじってない

キーペアは　picture01front_furuya.pem → furuya アカウントの ppk と同じ名前

　　IP は　********

サーバ設定

キーペアを利用し ec2-user でログイン後、furuya のアカウントを作り、秘密鍵でログインできるようにした
--
$sudo su -
#useradd -d /home/furuya -m furuya
#cd /home/furuya
#mkdir .ssh
#ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /home/furuya/.ssh/id_rsa
Enter passphrase (empty for no passphrase):　（パスフレーズはなしにした）
Enter same passphrase again:

#chown -R furuya .ssh
#chmod 700 .ssh
#mv .ssh/id_rsa.pub .ssh/authorized_keys
#chmod 600 .ssh/authorized_keys
#mv .ssh/id_rsa /home/ec2-user
#chmod 666 /home/ec2-user/id_rsa

picture01front_furuya.pem から PuTTYGen で picture01front_ec2-user.ppk を作成
（パスフレーズはなしにした）

WinSCP で picture01front_ec2-user.ppk を利用し ec2-user でログイン、id_rsa をダウンロード

PuTTYGen で id_rsa から picture01front_furuya.ppk を作成

#passwd furuya → PW:******** に設定
--

以上を kawagoe のアカウントに対しても行った

→→********に対して ID:furuya PW:********, ID:kawagoe PW:********でログインできるようになった

furuya と kawagoe が sudo できるようにした

sudoer を編集

ec2-user で
$sudo su -
#visudo

root　　ALL=(ALL)　　ALL　の下に
furuya　　ALL=(ALL)　　ALL
kawagoe　　ALL=(ALL)　　ALL　を追加

nginx の設定

EC2 のセキュリティグループのインバウンドルールに HTTP（ポート 80）を設定

以下を実行してインストール
$ sudo yum update
$ sudo yum install -y nginx
$ sudo chkconfig nginx on
$ sudo service nginx start

DocumentRoot は /usr/share/nginx/html/ になっている

設定ファイルは /etc/nginx/nginx.conf

Basic 認証のため .htpasswd を作成する
$ cd /usr/share/nginx/html
$ printf "********:$(openssl passwd -crypt ********)¥n" | sudo tee .htpasswd
これで ID:******** PW:******** になった

nginx に Basic 認証の設定を追記する
$ sudo vim /etc/nginx/nginx.conf

```
server {
    listen 80;
    server_name localhost;

    #charset koi8-r;

    #access_log /var/log/nginx/host.access.log main;

    auth_basic "basic authentication";
    auth_basic_user_file "/usr/share/nginx/html/.htpasswd";

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
    }
```

nginx を再起動し設定を適用させる
$ sudo service nginx reload

バックサーバ設定書
設定：古谷

参考
http://kazsoga.com/amazon-linux-python3-install/
http://symfoware.blog68.fc2.com/blog-entry-1412.html
http://qiita.com/moiwasaki/items/57f0f3382ca580c3b6d5
http://yakiniku.hatenablog.com/entry/2015/01/31/192757

Anaconda
Python3 のアンインストール（入ってた場合）
/etc/profile の最終行にあった PATH 設定文（$ export PATH=$PATH:/opt/ptyhon3/bin）を削除
Anaconda のインストール用.sh ファイルをダウンロードし実行
インストール先は /opt/anaconda3 を指定
$ export PATH="/opt/anaconda3/bin:$PATH" を /etc/profile に追加
pip でライブラリを再インストール
pyramid
arc-back を使うときは以下も同時に
$ python setup.py develop
$ initialize_arc-back_db development.ini
pit
$ sudo patch -u /opt/python3/lib/python3.4/site-packages/pit.py pit_py3.diff を忘れず
psycopg2

R
$ sudo yum install -y R git libxml2-devel openssl-devel libcurl-devel libjpeg-devel libpng-devel mysql-devel

PypeR
$ sudo pip install pyper

PostgreSQL をインストール
path を設定

postgresql-devel をインストール

psycopg2 をインストール