情報学部専用学内フォーラムサイトの構築

小澤 卓也 A)
静岡大学 技術部情報部門 A)

１．はじめに

静岡大学情報学部では毎年初夏に研究室配属が実施されている。教員と学生間で配属に関わる情報共有を行うため、情報学部専用の掲示板システムを利用していたが、システム更改に伴いこれまで利用していた掲示板システムが利用不可になった。

そこで代替システムとして、フォーラムサイト(掲示板システム) を新規構築した。Node.js を用い、既存の認証システムと連携しつつ、リアルタイムに投稿を閲覧できるサイトをフルスクラッチで構築し一新した。本稿では構築したシステムの概要・特徴的な機能の実装・運用経過について報告する。

２．システムの概要

２.１　Web サイトの設計

本システムは研究室配属期間中に、教員と学生間の双方向でコミュニケーションをとることを想定したシステムである。具体的な例として、教員から学生へ研究室説明会のお知らせをする場合や、学生の質問を受けるといった利用を想定した。利用者はブラウザからフォーラムサイトへアクセスし、教員からのお知らせを閲覧したり、コメントしたりできる (図 1)。

本システムではフォーラム・トピック・記事・コメントの 4 つの概念で構成される。研究室配属で利用する上ではこれらは次のように扱われる。まず、指導教員毎に「フォーラム」と呼ばれる領域が割当てられる。指導教員はフォーラムの中で、情報提供・コメントする場である「トピック」を自由に作成することができる。トピック内では、学生や他の教職員に共有したい「記事」を作成・投稿することで、他の学生や教職員が情報を閲覧することができる。投稿された記事に対しては、教職員・学生が自由にコメントを投稿できる (図 2)。

各ユーザはブラウザから本システムへアクセスし、これらの機能を利用できる(図 1)。閲覧したいフォーラムやトピックを左列から選択すると、中央列に投稿記事の一覧が表示される。更に記事を選択すると、その記事に対する各ユーザからのコメントが

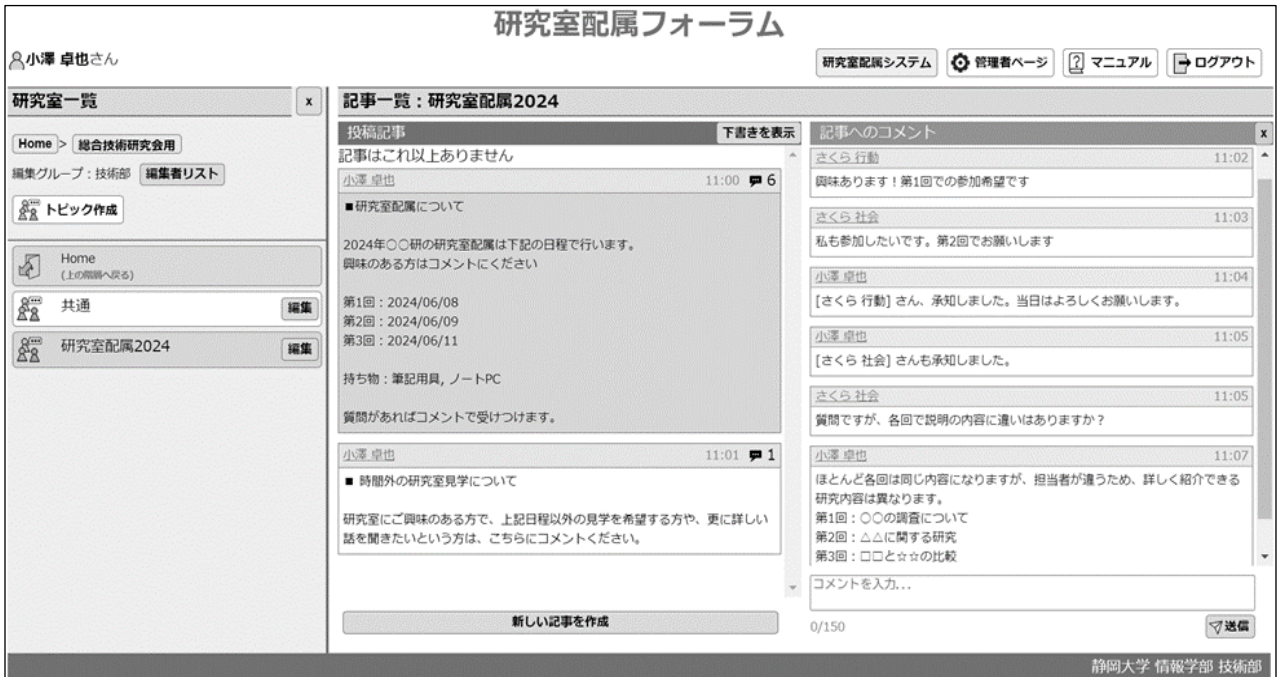

図１　本システム (研究室フォーラムサイト) メインページ

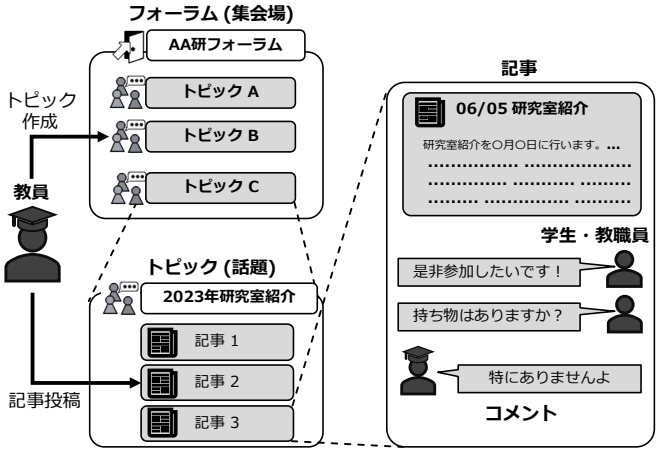

図 2　フォーラムサイトの概要

右列に表示される。特に、コメントや記事については、昨今のチャットコミュニケーションツールのように、最新の投稿を自動取得しながら投稿できる仕組みを実装した。

なお本システムへのアクセスはセキュリティ向上のため、学内ネットワークに接続している場合のみに限定した。本システムの利用者は利用目的からして情報学部構成員向けであるため、学外からの利用に制限をかける形とした。別の根拠としては、本システムとは別に研究室配属システム (全情報学部生の希望研究室を取りまとめるシステム) があり、こちらも学内専用のシステムであることから、それに合わせて本システムも学内専用とした。

2.2　システム全体像

本システムは大まかに、4 つのシステムを組み合わせて実現した (図 3)。1 つ目はクライアントとシステム間の通信路を暗号化するために Apache[1] を導入した。クライアントからの要求をアプリケーションサーバ (AP サーバ) に転送しつつ、AP サーバからの応答をクライアントに返す役割も担うよう、リバースプロキシとして稼働させた。

2 つ目はユーザ認証のための LDAP サーバである。本システムでは、記事やコメントを投稿したユーザの表示や、利用者を情報学部構成員に限定する目的のため、利用者の認証や氏名・連絡先といった個人情報を取得する機能が必要となる。これらの機能は既存の LDAP サーバと連携することで実現した。詳細は第 3 章で述べる。

3 つ目は、投稿データを保管するためのデータベース (DB) サーバである。こちらは MariaDB を利用し、必要な情報の整合性を保ちながら格納できるようテーブル設計した。データそのものへのアクセスは AP サーバを介してのみ可能とし、システムにログインしたユーザのみ AP サーバの API を介してアクセスできるようにした。詳細は第 4 章で述べる。

4 つ目は、本システムの中核となる AP サーバである。クライアントの要求に対して、DB や、認証サーバと連携してサービスを提供する API を実装した。後述するリアルタイムに投稿を配信する仕組みのため MQTT Broker と併せて実装した。詳細は第 5 章で述べる。

2.3　AP サーバのフレームワーク

AP サーバは Node.js を用いて実装した。Node.js はバックエンドだけでなくフロントエンドのためのツールとしても広く利用されている JavaScript の実行環境である[2]。すなわち、AP サーバ側のプログラムと Web ブラウザ上で動作するプログラムを、同じ JavaScript で記述できるため、開発者にとっては学

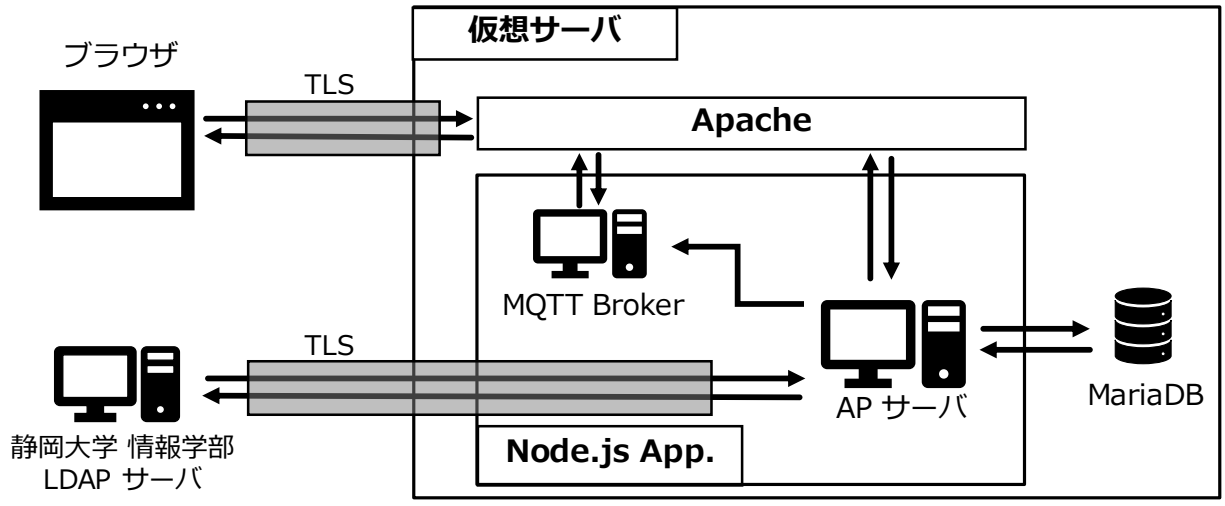

図 3　本システムの全体構成

習コストを抑えられる点や、フロントエンド・バックエンドを一貫して構築できる点で利点となる。また Node.js は、HTTP をやり取りするサーバとしてユーザからのリクエストを処理するなど、ネットワークの処理を得意としている[2] 。

本稿で開発したシステムにはAPIとして実装すべき機能が数多く存在する。これらを効率的に開発するために Express も併せて導入した。Express は Node.js 登場初期から広く利用されている Web フレームワークであり[3]、API を迅速かつ簡単に作成できる[4] 特徴をもつ。ネットワーク処理を得意とする Node.js に Express を併用することで、サーバサイドの処理を効率的に構築できる。本稿の AP サーバでは、Express の持つ機能のうち、ルーティングとミドルウェア、包括的エラーハンドリングを利用した。

クライアントからのリクエストを受け取った AP サーバは、URI に応じてあらかじめ設定したミドルウェアを実行する。Express では、この URI とミドルウェアを紐づける、ルーティング機能が実現されている。この機能を用いて例えば「とある記事へのコメントを取得したい」「フォーラム一覧を取得したい」「記事を投稿したい」といったクライアントからの要求それぞれに対し、AP サーバ上で実行するプログラムを分割して実装・管理した。

また、包括的エラーハンドリングの仕組みを用い、各ミドルウェアの実行中にエラーが発生した際、クライアントに対して共通のエラー処理を実現した。包括的エラーハンドリングの設定を自身で行わない場合、組み込まれたデフォルトのエラーハンドラーが呼び出される[5]。デフォルトのエラーハンドラーではスタックトレースなどの、アプリケーションに関する詳細なエラーがクライアントに返却されてしまう。これらの情報は攻撃に利用される可能性がある。そこで全ルーティングで共通のエラー処理として「何らかのエラーが発生した」旨のみをクライアントに通知するように、包括的エラーハンドリングを実装しサイトの安全性を高めた。

3．既存の認証システムとの連携

3.1　情報学部の認証システム

本学情報学部では各種情報サービスの提供にあたり、認証基盤を整備している。認証基盤の中にはユーザ情報の管理や認証サービスの提供をする LDAP サーバが稼働している。本稿で構築するサービスはこれら既存の認証システムを活用して、ログイン機能やユーザ情報を取得するよう設計した。

これらの機能を既存のLDAPサーバと連携するメリットは利用者・管理者ともに存在する。利用者にとっては、情報学部のシステムを共通の認証情報で利用することができるため、認証情報の管理コストを下げられる。将来的にはシングルサインオン(SSO)の実現により、利便性の向上も見込める。

また管理者にとっては、パスワードやユーザ情報等の重要な情報を、構築するシステムに持たせる必要が無くなる点が大きな利点となる。ユーザ情報を LDAP サーバの情報に一任することでデータを一元管理し、管理コストを下げられる。また、重要な情報を構築するシステムに持たせないことから、AP サーバにおけるセキュリティ向上も見込める。

3.2　AP サーバと LDAP 認証の連携

ログイン時にはブラウザから認証情報として、ユ

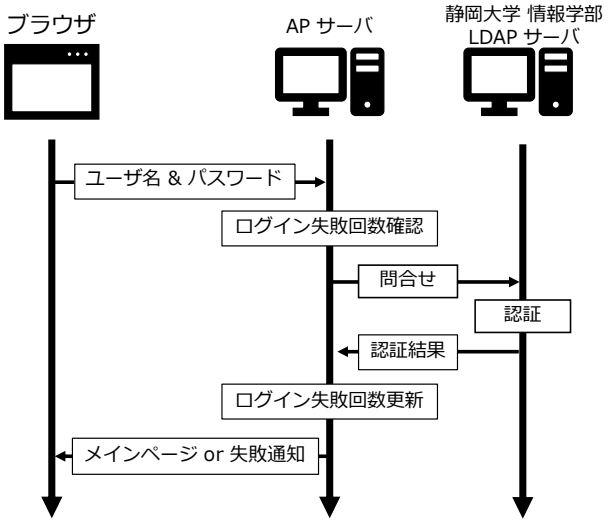

図 4　ログイン時の処理

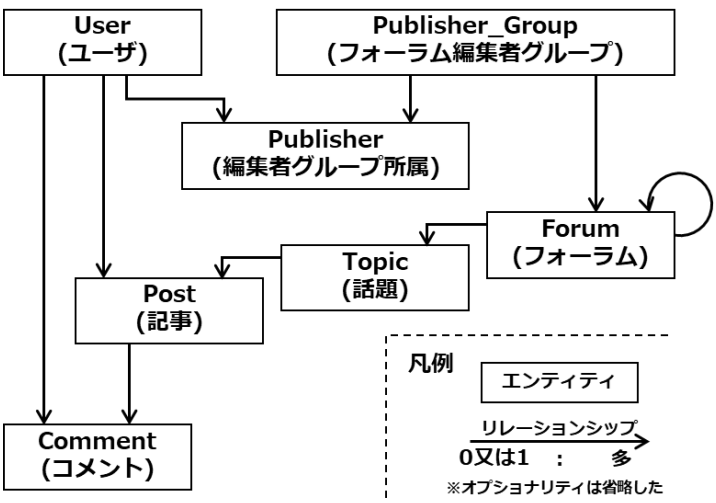

図 5　本システムの概念データモデル

ーザ名とパスワードの組を受け取り、AP サーバ・LDAP サーバが連携してログイン可否の判定を行う(図 4)。ユーザは本システムを利用する際、ログインページにアクセスして認証情報を AP サーバへ送信する。認証情報を受け取った AP サーバは LDAP サーバと暗号化した通信路を確保し、代理で認証サーバへ認証要求を送る。認証結果を基にメインページ又は失敗通知を利用者へ返す。

上記の方法を単純に実装するとブルートフォースアタックと呼ばれる、パスワードの総当たりによる攻撃で破られるセキュリティリスクを抱えてしまう。この対策として、各ユーザの連続ログイン失敗回数を本システム上でカウントし、認証そのものの実施可否を本システム上で判断する。連続ログイン失敗回数が一定値を超えると、認証要求を認証サーバに問い合わせることなくログイン失敗として扱う。これにより総当たり攻撃への耐性を確保し、セキュリティリスクを低減させた。

４．データベース設計

４.１　AP サーバと DB の連携

投稿した記事やコメントなど、本システムの稼働で記憶する必要のある情報は DB で管理する。DB 管理システムとして今回は MariaDB を採用した。AP サーバが稼働する仮想環境上に DB を構築し、AP サーバからの要求に応じて必要な情報を参照・追加・削除する。これらの DB に対する操作クエリは予め MariaDB 上にストアドプロシージャとして定義し、AP サーバを介してのみ実行できるようにした。この理由は以下の 3 点である。

1 点目は、複数の SQL 文からなる手続きを 1 回の呼び出しで実行できる点である[6]。複数の SQL 文から成る一連の処理が AP サーバと DB 間において 1 往復で完結するため、スループットが向上する。

2 点目は、システム全体で共通な処理をプロシージャとして格納しておくことによって、処理の標準化を行うことができる点である[6]。利用頻度の高いクエリをプロシージャとして定義することで、システム全体で開発・管理がしやすく、結果として信頼性の高いシステムを組み上げることが出来る。

3 点目は、機密性の高いデータに対する処理を特定プロシージャ呼び出しに限定することによって、セキュリティを向上させることが出来る点である[6]。具体的に本システムでは、ユーザのログイン状況を DB で管理しているが、その情報に対するアクセスをストアドプロシージャで包む設計とした。これにより、不必要に機密性の高い情報にアクセスしてしまうリスクを抑えた。

４.２　テーブル設計

本システムで必要な情報を DB で管理する際、データの整合性を担保するようにテーブルを設計し

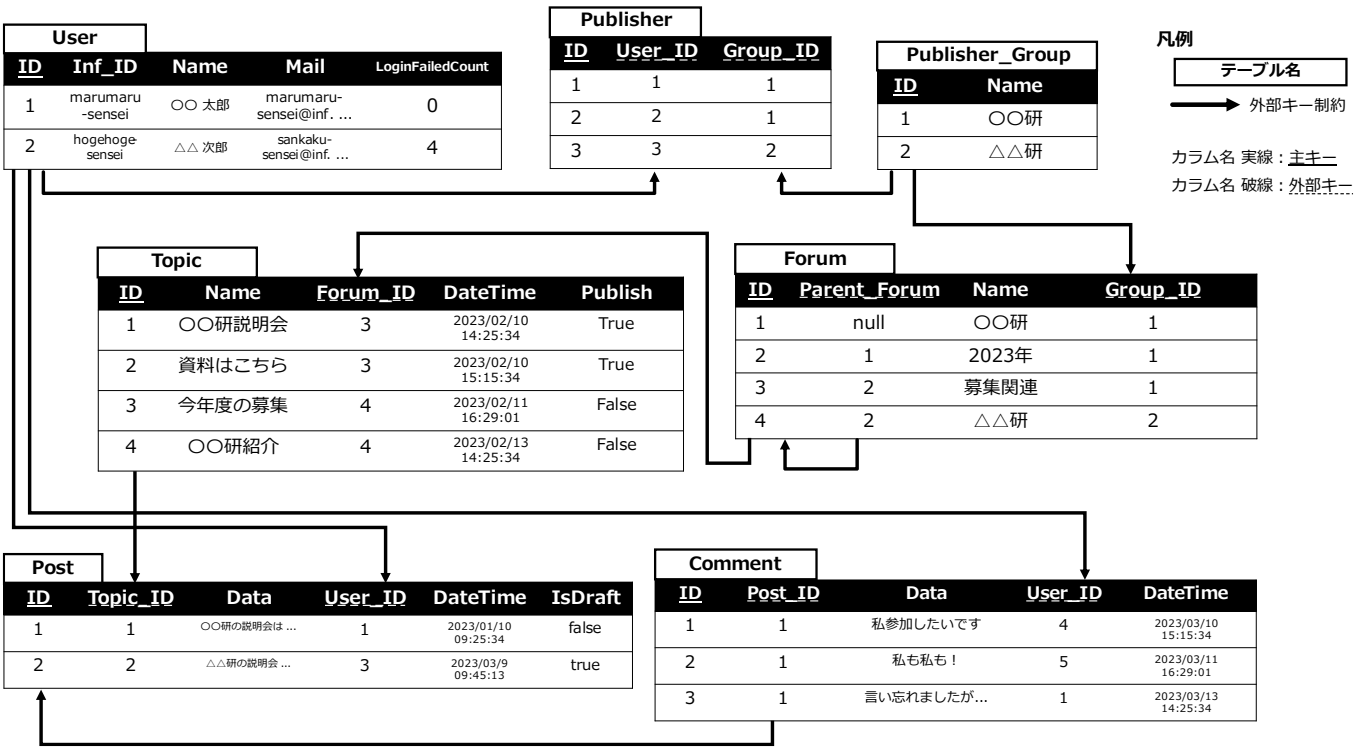

図６　本システムの DB テーブル設計

た。具体的には、システムで必要な要素を洗い出し、概念データモデル(図 5)とテーブル定義(図 6)を設計した。特にテーブルは第 3 正規形にし、今後の機能拡張にも柔軟に対応できる設計とした。

設計のうち、特筆すべきは以下の 2 点である。1 点目はフォーラムを木構造のような階層構造を持てるようにした点である。すなわち、フォーラムの下にサブフォーラムが複数存在し、そのサブフォーラム以下に更にサブフォーラムが存在するという関係を、フォーラムの親子関係をもって表現した。具体的には、フォーラムの階層構造を表現するため、Forum テーブル内に自身の親フォーラムへの参照を示す、自己参照外部キー Parent_Forum を持たせた。この実装方法は隣接リストモデルと呼ばれる[7]。

2 点目は、フォーラムの編集・記事投稿権限をグループ単位で付与できるようにした点である。この実装自体は極めてシンプルに、Forum テーブルにユーザグループへの外部参照を持たせることで実現した。また、あるグループに対しては複数のユーザが所属でき、逆にあるユーザは複数のユーザに所属できる、という性質から、User と Publisher_Group 間には多対多の関係がある。したがって、この 2 つのエンティティ間をつなぐ関連実体 Publisher を定義し、上記の多対多の関係を表現した。

５．コメント・記事のリアルタイム更新

一般的な Web サイトにおいて、サーバ上の最新情報を利用者が得ようとした場合、利用者側から能動的にサーバへ要求を送る必要がある。この仕組みの上で常に最新の記事やコメントを受け取りたい場合、新規投稿が無いか定期的にサーバに問合せる「ポーリング」と呼ばれる実装が考えられる。しかしこの実装は、大勢の利用者が同時アクセスした場合に、システムに対する負荷が増大する問題がある。

具体的には DB の検索処理やネットワークの負荷が増大する。通常、最新情報をクライアントが要求した際はまず、ブラウザから AP サーバに記事やコメントの取得要求を送信する。要求を受け取った AP サーバは DB に対して検索要求を発行し、テーブル内の検索を実行する。最後に、検索結果をクライアントに返す処理が実行される。ポーリング方式ではデータが更新されていない場合も含め、これらの処理が定期的に実行される。このとき特に DB サーバとネットワークに大きな負荷がかかってしまう。また、リアルタイム性を重視するならば、ポーリングの時間間隔を短く設定することになるが、要求間隔を短くするにつれサーバの処理は増大する。また同時アクセスするクライアント数にも比例し、AP サーバや DB の負荷が増大する問題がある。

そこで本システムではその方法は採用せず、「Message Queuing Telemetry Transport (MQTT)」と呼ばれるプロトコルを利用することとした。MQTT は MQTT Broker と MQTT Client の 2 種類の要素で構成される。MQTT Client は受信したいメッセージ内容 (Topic) を予め MQTT Broker に通知 (Subscribe) しておく。この状態で、別のクライアントから Topic に関するメッセージを配信 (Publish) すると、MQTT Broker は Topic を購読している MQTT Client に対してメッセージを一斉配信する。

このプロトコルを用い、最新記事やコメントをサーバ側から能動的に配信する仕組みを実現した (図 7)。投稿を受け取った AP サーバは DB と連携して投稿可否の検証を行い、問題なければ DB へ投稿内容を保存する。保存に成功したのち、MQTT Broker

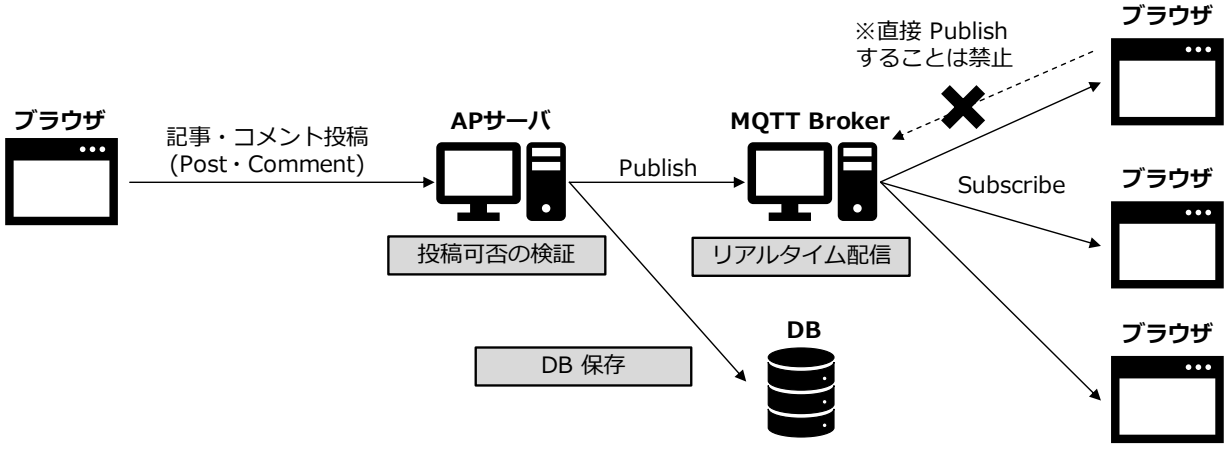

図 7　記事・コメント投稿時の処理

に対して投稿内容を Publish する。投稿内容を受け取った MQTT Broker は、コンテンツ閲覧中の利用者に対して投稿内容を一斉配送する。

この手法の利点は、記事やコメントが投稿されたタイミングで各クライアントへ 1 回のみデータを送信すればよく、必要最低限の通信で全クライアントへ配信できる点である。また当該トピック・記事を閲覧中のクライアントに対して MQTT Broker から能動的にデータを送付するため、定期的に DB を検索する処理が不要となる。したがって DB への負荷も低減することができた。

なお一般的には、MQTT Broker に対して MQTT Client は自由に Publish できてしまう。そのままこのシステムに適用すると、各クライアントが自由にデータを配信できることになり、セキュリティ的に問題となってしまう。特に、記事の投稿に関しては、フォーラムを管理する教職員のみに限定したい要件がある。そこで本システムでは、AP サーバから直接要求された Publish 以外は無視するように実装した。Publish するための窓口を AP サーバに限定して送信データを検証することで、無関係なユーザが勝手に記事を投稿できないようにした。

なお本システムでは MQTT Broker ライブラリとして Aedes[8]を採用した。こちらのライブラリは Node.js 向けに作られたライブラリであり、AP サーバに容易に組み込むことができた。

６．運用

2024 年 5 月下旬より本システムの運用を開始した。実際に当学部の行動情報学科の研究室配属期間において、各研究室と学生間の情報共有ツールとして 2 か月間運用した。

運用にあたり、学生・教員向けにそれぞれマニュアルを完備した。運用期間中に利用方法に関する初歩的な問い合わせはなかったことから、マニュアルについては一定の効果があったものと思われる。

また運用中に教員から、学外からも閲覧できるようにしてほしいという要望があった。そこで既存の情報学部のシステムとも連携し、ユーザ認証のもと学外からも閲覧できるようにすることで本システムの利便性を高めた。検証の際は学内展開する際と同等の機能が利用できるかテストを実施し、問題ないことを確認した上で学外からのアクセスを可能にした。

７．まとめ・今後の展望

本稿では研究室配属向けのフォーラムサイトを構築した。新規システムの開発・提供を考えたとき、ユーザからのアクセスを安全に捌ききる必要があった。セキュリティ上のリスクを想定して事前に可能な限り対策することや、処理効率の良い方法を採用する工夫で解決した。

本項で述べた機能のうち特に、認証機能やサーバから能動的にクライアントへ情報を渡す仕組みは、複雑な Web サービスを提供するうえで有用な仕組みである。今後他の Web サービスを開発する際には、これらの機能の流用も視野に入れ、開発期間の短縮と品質向上を両立するようにしたい。そのために、本稿で実装した機能を使いやすいライブラリとしてまとめ、今後の開発に役立てていきたい。

参考文献

[1] Apache Software Foundation. “Apache HTTP Server Project”. https://httpd.apache.org. (参照 2024-11-20)

[2] 伊藤 康太, 実践 Node.js 入門 -基礎・開発・運用, 技術評論社, 2023, p.2.

[3] 伊藤 康太, 前掲文献, p.155.

[4] OpenJS Foundation. “Express - Node.js web application framework”. https://expressjs.com. (参照 2024-10-3).

[5] 伊藤 康太, 前掲文献, p.166-p.167.

[6] アイテック 情報技術教育研究所, データベース技術, 2001, p116-117.

[7] ミック, 達人に学ぶ DB 設計 徹底指南書 ~初級者で終わりたくないあなたへ, 株式会社 翔泳社, 2022, p.265.

[8] GitHub. “Aedes - Barebone MQTT broker that can run on any stream server”. https://github.com/moscajs/aedes. (参照 2024-11-20).

謝辞

本システムの構築・本稿執筆にあたりまして、多くのご指導・アドバイスをいただきました、静岡大学技術部情報部門 水野匠 様、柴田頼紀 様 に感謝申し上げます。