2022年度 卒業論文

ポッドキャストサブスクリプションの開発

大阪産業大学 デザイン工学部 情報システム学科
情報教育システム研究室

19H104 門口大祐

ポッドキャストサブスクリプションの開発

19H104 門口大祐

1　はじめに

近年音声コンテンツへの注目が高まっている。その理由として別の事柄と並行してコンテンツを聞く「ながら聞き」が可能となっており、手軽に情報収集できることが強みとなっている。特に音声コンテンツの一つであるポッドキャストへの需要は高い。世界のポッドキャストのリスナー数は 2020 年には 3 億 3 千人いるとされる [1]。2021 年 6 月に Apple 社のポッドキャストの配信サービスおよび再生アプリケーションである ApplePodcasts のみでしか聴くことのできないサブスクリプションサービスを開始し、ポッドキャストの収益化の流れが始まった。しかしポッドキャストは元来 RSS [*1]という文書フォーマットに更新情報を記述したものを用いて発信しており、利用者はそれを取得して各々の所持している再生アプリケーション内で聞く事ができた。よって ApplePodcasts のサブスクリプションサービスは RSS を用いていないため他の再生アプリケーションで聞くことができず、標準的方式を満たしていないためポッドキャストと呼べないと考える。

2　目的

本研究の目的は RSS に記述されているメディアファイルが取得される際にユーザ認証を追加し、ポッドキャストの標準的方式を用いたポッドキャストサブスクリプションのシステム開発である。

3　システム

本研究では RSS に記述されているメディアファイルが置かれているサーバに RFC7519 標準規格である JWT[*2]を用いて認証情報を送受信し、ユーザ認証を行う WebAPI を作成した。WebAPI にはログイン・JWT 発行 API とユーザ認証 API がある。ログイン・JWT 発行 API はログインしようとするユーザ情報がシステムに既存のユーザ情報と一致するかを調べ、一致した場合に JWT の発行を行う。ユーザ認証 API はユーザに発行された JWT の検証を行いユーザがサブスクリプション限定のポッドキャストを再生できるかどうかの認証を行う。

4　結果

本システムが正常に機能するかを検証するために、本システムに対応したユーザ情報と JWT を授受するポッドキャスト再生アプリケーションを作成した。本システムにて発行した URL を記述した RSS を読み込ませ動作の検証を行った。想定通りに認証が働きポッドキャストの標準的方式を満たしたサブスクリプションのシステムを開発できた。しかし既存のアプリケーション上では認証に対応していないため動作しないのが課題となった。

5　まとめ

現状のポッドキャストのサブスクリプションサービスがポッドキャストの標準的方式を満たしておらず、本研究ではそれを満たしたサブスクリプションのシステム開発を行った。結果、本システムに対応したアプリケーション上での再生には成功した。しかし既存のアプリケーションでは本システムの認証には対応していないため認証方式の標準化が今後の課題である。

参考文献

[1] Global podcast listener forecast 2021–2025 - insider intelligence trends, forecasts & statistics. https://www.insiderintelligence.com/content/global-podcast-listener-forecast-2021-2025.

[*1] Really Simple Syndication の略称。ヴァージョンによって名称が変更されているが、ポッドキャストの場合はヴァージョン 2.0 を用いるため前述した名称で表記する。

[*2] JSONWebToken の略称。

目次

1　はじめに

近年音声コンテンツへの注目が高まっている。その理由として視覚情報に重きをおいたコンテンツは視聴することにリソースを集中させる必要がある事に対し、音声コンテンツは場所や時間に囚われずに済むことが挙げられる。よって別の事柄と並行してコンテンツを聞く「ながら聞き」が可能となっており、通勤時間などにも手軽に情報収集できることが強みとなっている。特に音声コンテンツの一つであるポッドキャストへの需要は高く、新聞社やテレビ局がニュース配信を行う際にも利用されている。世界のポッドキャストのリスナー数は 2019 年に約 2 億 7 千万人、2020 年には 3 億 3 千人と推移している。[1] この予測通りにリスナーが増加すると 2024 年にリスナーは約 5 億人になると試算される。この推移と予測値を**図 1** に示す。日本国内向けに 2020 年度末に行われた調査 [2] から日本国内ポッドキャストリスナーは 1123 万 7 千人と推計されるほどとなっており国民の 1 割が利用していることから注目の高さが伺える。

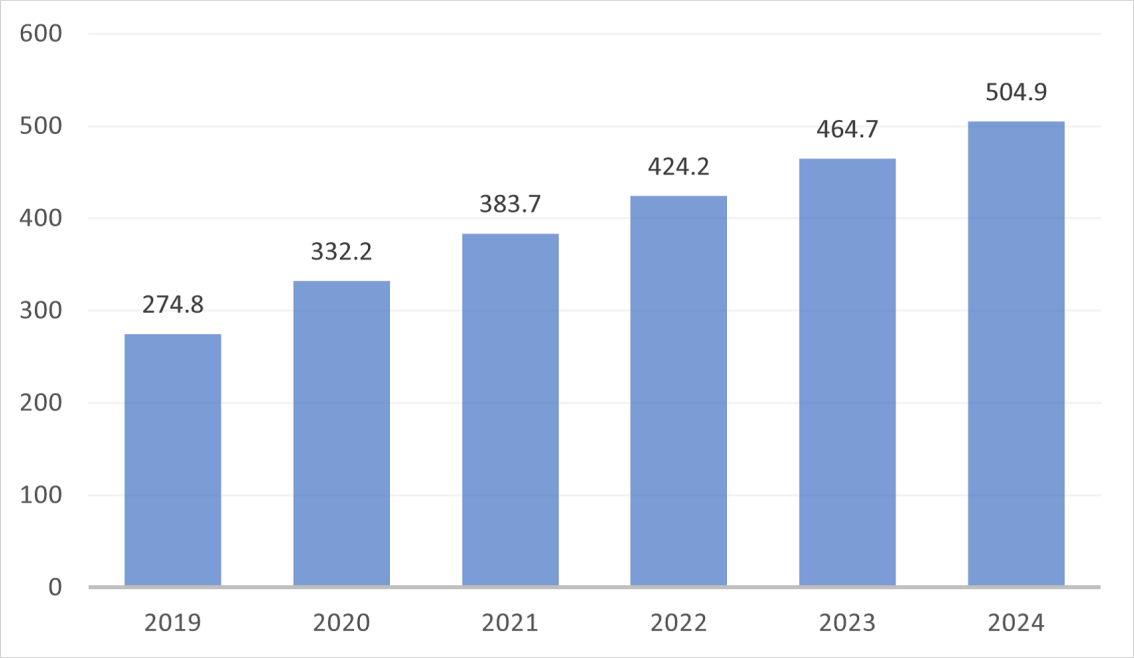


図 1　全世界のポッドキャスト利用者数の推移と予測
eMarketer 調べ。縦軸はリスナー数（単位:百万人）、横軸には西暦を示す。サンプリングされた国には、アルゼンチン、オーストラリア、ブラジル、カナダ、中国、デンマーク、フィンランド、フランス、ドイツ、イタリア、日本、メキシコ、ノルウェー、韓国、スペイン、スウェーデン、英国、および米国が含まれる。 2021 年から 2024 年までは予測値。

そして、ここまでリスナーが増えてくるともちろんコンテンツの配信者側の利益追求にも話は繋がってくる。2021 年 6 月に Apple 社の提供するポッドキャストの配信サービスおよび再生アプリケーションである Apple Podcasts が購買した利用者だけにポッドキャスト番組を提供するサブスクリプションサービスを開始し、ポッドキャストの収益化の流れが始まった。多くのポッドキャスト配信サービスは RSS という文書フォーマットに更新情報を記述したものを用いて発信しており、利用者はそれを取得して各々の所持している再生アプリケーション内で聞く事ができた。しかし Apple Podcasts のサブスクリプションサービスは RSS を用いていないため他の再生アプリケーションで聞くことができない。よって利用者としては不便な点が存在する。

第 2 章では本研究の目的について述べる。第 3 章ではポッドキャストと既存のポッドキャストに類似したサブスクリプションサービスの問題点を述べる。第 4 章では本研究で開発した認証システムについて述べる。第 5 章

には本研究で行った検証について述べる。第 6 章には本研究で開発したシステムの考察についてまとめる。第 7 章には研究の成果について述べる。

2　目的

現状のポッドキャストのサブスクリプションサービスの問題点として、その実現に RSS を用いないためリスナーが各々に所持している再生アプリケーションでの効率的な利用ができない点が挙げられる。配信サービスの違いはあるが RSS を用いて更新情報を配信していれば一つの再生アプリケーション内に更新情報を集約することが可能であるのがポッドキャストの利点である。これをサブスクリプションサービスにも対応させることで元来の利点を活かし、サブスクリプションサービスを利用しているリスナーの不便を解決できると考えた。本研究の目的は RSS に記述されているメディアファイルを取得する際にユーザ認証を追加し、ポッドキャストの標準的方式を用いたポッドキャストサブスクリプションサービスのシステムの開発である。

3　ポッドキャスト

本セクションではポッドキャストの概要とポッドキャストの配信フォーマットである RSS の概要およびポッドキャストに類似したサブスクリプションサービスについて述べる。

3.1　ポッドキャスト

ポッドキャストとは、Web サーバ上に音声データや動画データなどのメディアファイルをアップロードし、RSS を用いて Web 上に公開するインターネットラヂオ・インターネットテレビの一種である。語源は Apple 社のポータブルマルチメディアプレイヤーの「iPod（アイポッド）」と放送を意味する「broadcast（ブロードキャスト）」を組み合わせた造語である。しかしポッドキャストの出自自体は名前がつけられる前からインターネットラヂオの一種として存在していた為 Apple 社とは無関係である。一般的にポッドキャスト（Podcast) という言葉が前述したデータファイルを Web に公開する仕組み自体を示し、ポッドキャスティング（Podcasting) はこの仕組みを用いてインターネットラヂオを配信するという行為を示すことに使われる。ポッドキャストはメディアファイルをダウンロードして聴くことが可能であり五感リソースも聴力だけを必要とするので、場所や時間を問わずに利用できる利用効率の良いコンテンツであると言える。代表的なポッドキャストアプリケーションとして、Apple Podcasts、Google Podcasts、Anchor、Spotify、Amazon music、SoundCloud 等が挙げられる。それらのアプリケーションのアイコンを**図 2** に示す。

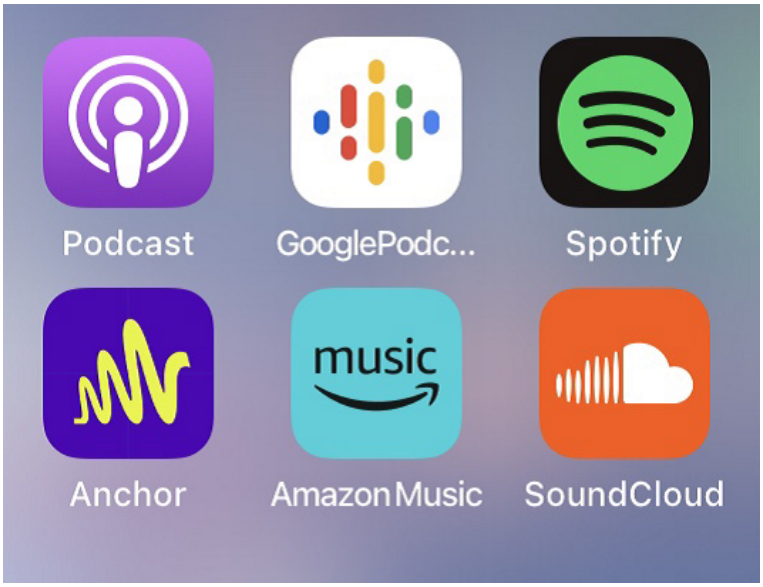


図 2　ポッドキャストが視聴できるアプリケーションのアイコン
左上から時計回りにそれぞれ Apple 社製ポッドキャストアプリケーション ApplePodcasts(Apple 社製のオペレーティングシステム上では Podcast と表記される)、Google 社製ポッドキャストアプリケーション「GooglePodcasts」、SpotyfyTechnology 社製音楽ストリーミングアプリケーション「Spotify」、SoundCloud Limited 社製音声ファイル共有アプリケーション「SoundCloud」、Amazon 社製ポッドキャスト音楽ストリーミングアプリケーション「AmazonMusic」、SpotyfyTechnology 社製ポッドキャストアプリケーション「Anchor」

3.2 RSS とは

RSS とは Really Simple Syndication[*1]の略である。Web サイトの更新情報を配信するための文書フォーマットである。XML ベースのマークアップ言語で記述し、更新された WEB サイトのタイトル、URL、更新日時、要約などを記述することが出来る。その記述例を**図 3** に示す。こうして配信されたデータを「RSS フィード」または単に「フィード」という。この RSS をポッドキャスト再生アプリケーションに登録しておくことで、アプリケーションが定期的に RSS フィードを取得する。よって複数個 RSS を登録しておくことで、色々な配信サイトや配信者のポッドキャスト番組を取得し表示するので、更新を確認するために各サイトを閲覧する必要が無くなるのが利点である。

```
            <item>
                    <title><![CDATA[マインスイーパ]]></title>
                    <description><![CDATA[<p>地雷原</p>
]]></description>
                    <link>https://anchor.fm/owata-otawa/episodes/ep-e1a01si</link>
                    <guid isPermaLink="false">6ea03526-4033-4306-a025-0d838a31d54d</guid>
                    <dc:creator><![CDATA[Jonathan]]></dc:creator>
                    <pubDate>Tue, 09 Nov 2021 07:42:46 GMT</pubDate>
                    <enclosure url="https://anchor.fm/s/6f5dc464/podcast/play/43042130/https%3A154.m4a" length="1485361" type="audio/x-m4a"/>
                    <itunes:summary>&lt;p&gt;地雷原&lt;/p&gt;
</itunes:summary>
                    <itunes:explicit>No</itunes:explicit>
                    <itunes:duration>00:01:31</itunes:duration>
                    <itunes:image href="https://d3t3ozftmdmh3i.cloudfront.net/production/podcast_uploaded400/18584161/18584161-1634015582288-d130ab54449f6.jpg"/>
                    <itunes:episodeType>full</itunes:episodeType>
            </item>
```

図 3　SpotyfyTechnology 社製ポッドキャストアプリケーション「Anchor」で生成された RSS の記述例。item 要素の階層にある title 要素にタイトル、description 要素に要約や説明、pubDate 要素に更新日時、enclosure 要素にメディアファイルの URL、音声ファイルの秒数、ファイル形式が記述されている。

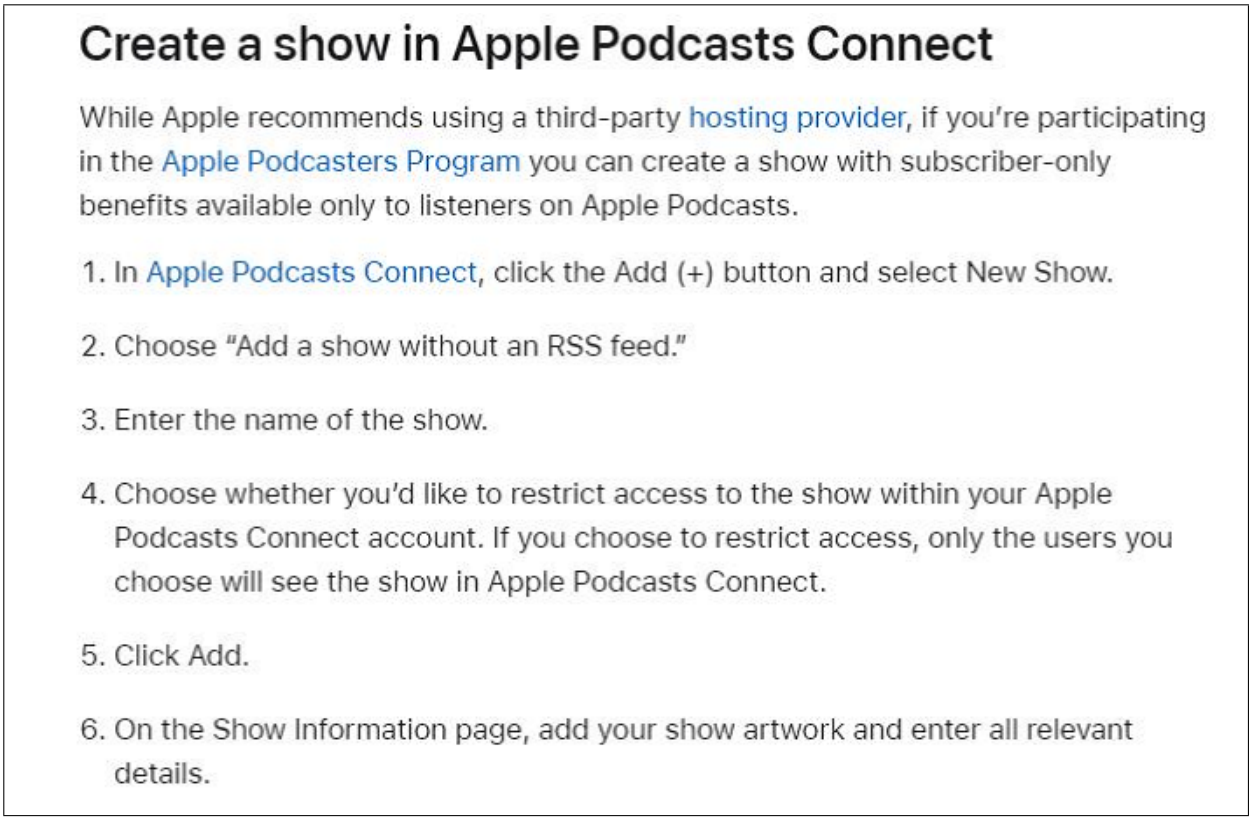
3.3 ポッドキャストに類似したサブスクリプションサービスの現状と問題

ポッドキャストに類似したサブスクリプションサービスとして ApplePodcasts と Substack が挙げられる。しかしこれらのサービスは後述する問題を抱えていると考え、現状ではポッドキャストとは呼べないサービスとなっている。これらのサービスの問題点を解消するために本研究を進める。

3.3.1 ApplePodcasts サブスクリプションサービス「ApplePodcastersProgram」の問題点

ApplePodcasts のサブスクリプションサービスの問題は利用する場合に RSS を用いることができない点である。これは ApplePodcasts サブスクリプションサービスである「ApplePodcastersProgram」のガイドからも確認できる。ガイドの当該部分を**図 4** に示す。よって他のアプリケーションを利用しているリスナーは ApplePodcasts のサブスクリプションされた番組を聞くのに ApplePodcasts を利用する必要があり、ポッドキャストの利点が失われてしまう。サブスクリプション番組を利用する際の ApplePodcasts の画面を**図 5** に示す。加え、前述した通り RSS を用いて配信するのがポッドキャストであり、用いないのであればポッドキャストと呼ぶことができないと考える。

[*1] ヴァージョンによって名称が変更されているが、ポッドキャストの場合はヴァージョン 2.0 を用いるため前述した名称で表記する



Create a show in Apple Podcasts Connect

While Apple recommends using a third-party hosting provider, if you're participating in the Apple Podcasters Program you can create a show with subscriber-only benefits available only to listeners on Apple Podcasts.

1. In Apple Podcasts Connect, click the Add (+) button and select New Show.
2. Choose "Add a show without an RSS feed."
3. Enter the name of the show.
4. Choose whether you'd like to restrict access to the show within your Apple Podcasts Connect account. If you choose to restrict access, only the users you choose will see the show in Apple Podcasts Connect.
5. Click Add.
6. On the Show Information page, add your show artwork and enter all relevant details.

図 4　ApplePodcastersProgram ガイド [3] の一部。2 にてサブスクリプションサービスを適用する場合 RSS を利用しないでポッドキャスト番組を登録する必要があることが明記されている。

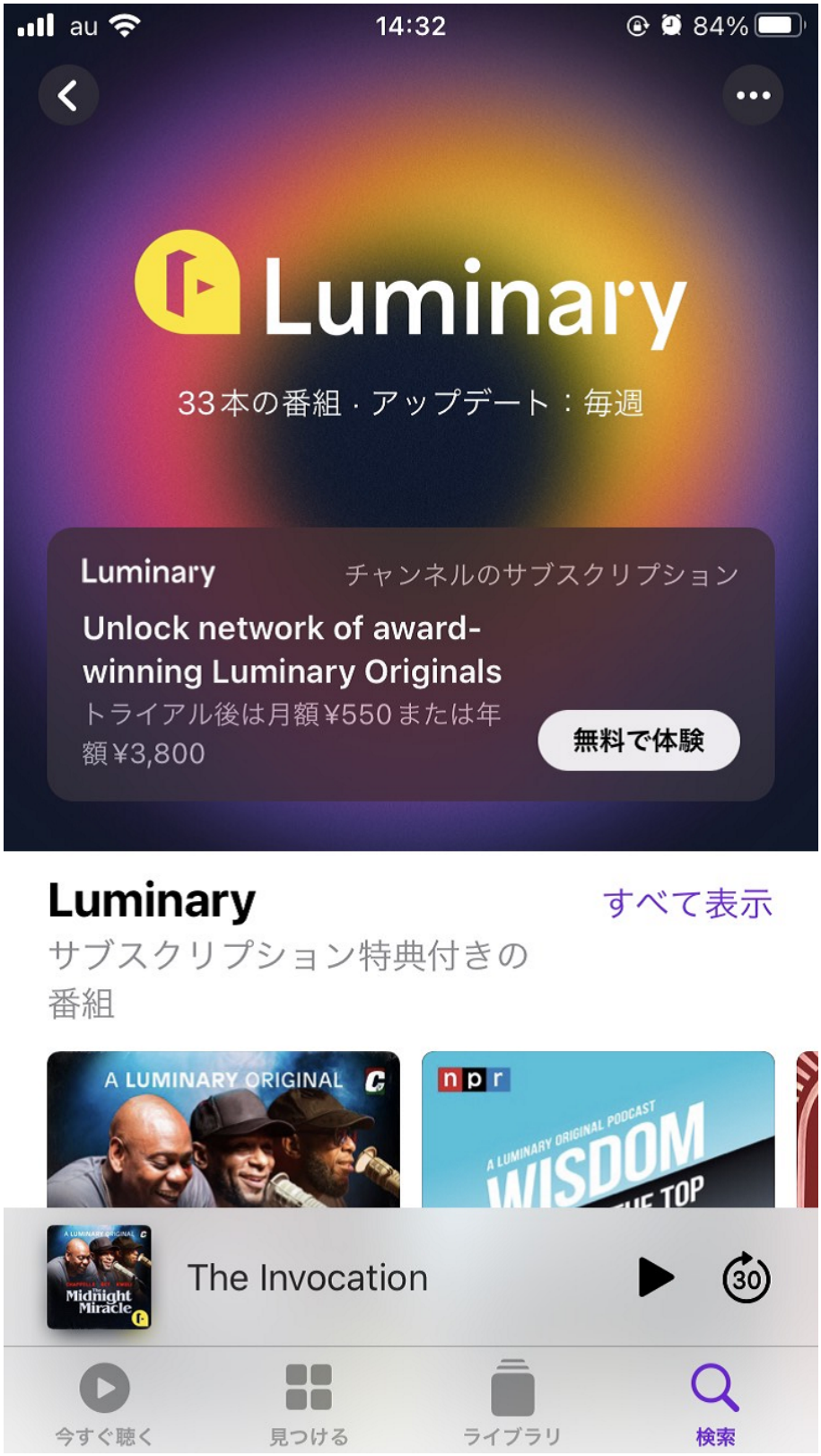


図 5　Apple 社のマルチメディアプレイヤー「iTunes」でサブスクリプションサービス「ApplePodcasters-Program」を利用している番組を利用する際の画面。RSS を用いていないため Apple 社製ポッドキャストアプリケーション「ApplePodcasts」でしか再生することができない。よってポッドキャストの標準的方式を満たしていないためポッドキャストと呼ぶことができないと考える。

3.3.2 サブスクリプションニュースレターサービス「Substack」の問題点

Substack のサブスクリプションサービスはリスナーごとにユニークな RSS を生成することでポッドキャストの配信を行っていることが確認できる。それが明記されている利用ガイドを**図 6** に示す。一見こちらには問題がないように感じるが、RSS に記述されている内容に問題が存在する。それは記述されているメディアファイルの URL にユーザ認証がない点である。よって URL 自体を何かしらの方法で取得することできれば誰であってもメディアファイルにアクセスし音声を聞くことができる。

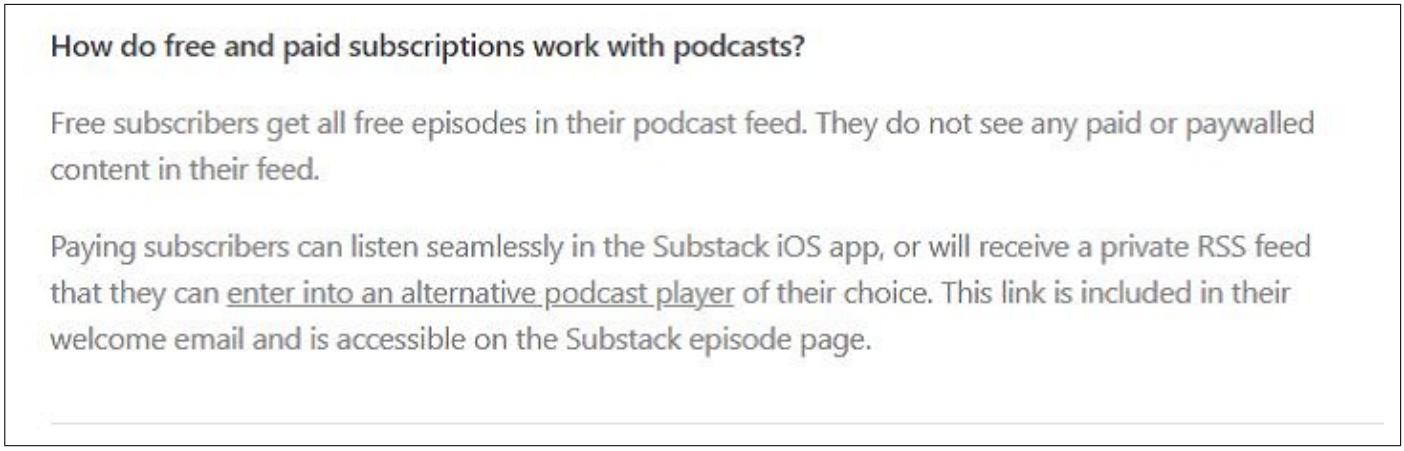


図 6　Substack の利用ガイド [4] の一部。4 行目からユーザごとに RSS を生成することが明記されている。

4　システムの概要

本研究は、RSS に記述されているメディアファイルが置かれているサーバにユーザ認証 WebAPI を作成し、JSONWebToken（以下 JWT とする。）を用いてユーザ認証を行う。これにより JWT を用いてユーザ情報を識別することで、特定のユーザだけがポッドキャストを聞く事の出来る RSS を用いたサブスクリプションサービスを展開できると考えた。本システムの概要図を**図 7** に示す。

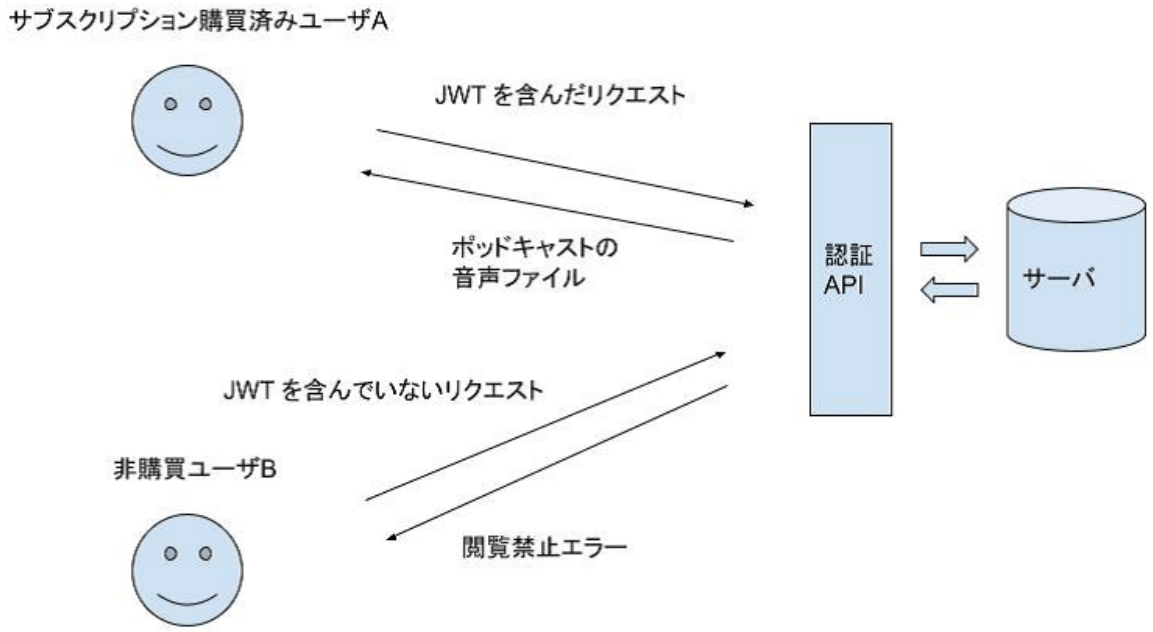


図 7　認証システムの概要図。ユーザ A はサブスクリプションを購買しているため、既に発行された JWT を含んだリクエストをし、認証されポッドキャストの音声ファイルがレスポンスされている。ユーザ B はサブスクリプションを購買していないため、JWT を含んでいないリクエストをしているのでエラーを返信されている。

4.1　JWT

JWT [5] は JSON 形式で記述された認証情報を安全に送受信できるように符号化したトークンを生成する仕組みを規定した RFC7519 標準規格である。トークンはサーバとクライアント間で送受信され、「秘密鍵」によって復合されることで記述されたデータを読み取ることができる。「公開鍵」を利用してトークンの正当性を確認し、改竄等が行われていないことを確認できることが特徴である。「ヘッダー（Header）」、「ペイロード（Payload）」、「署名（Signature）」の 3 つの要素で構成されている。これらを Base64URL 方式で符号化し、「.（ピリオド）」で連結したものがトークンとなる。ヘッダーはトークンや署名のアルゴリズム方式の情報を格納する部分である。ペイロードは伝達される情報の本体であり、システムの必要となる任意のデータを格納する部分である。署名はトークンの作成者の本人証明に用いられ、符号化されたヘッダーとペイロード部分を連結し、秘密鍵と共に署名アルゴリズムによって生成されたデータを格納する部分である。符号化されたトークンと復号後のヘッダーとペイロードと署名の例を**図 8** に示す。

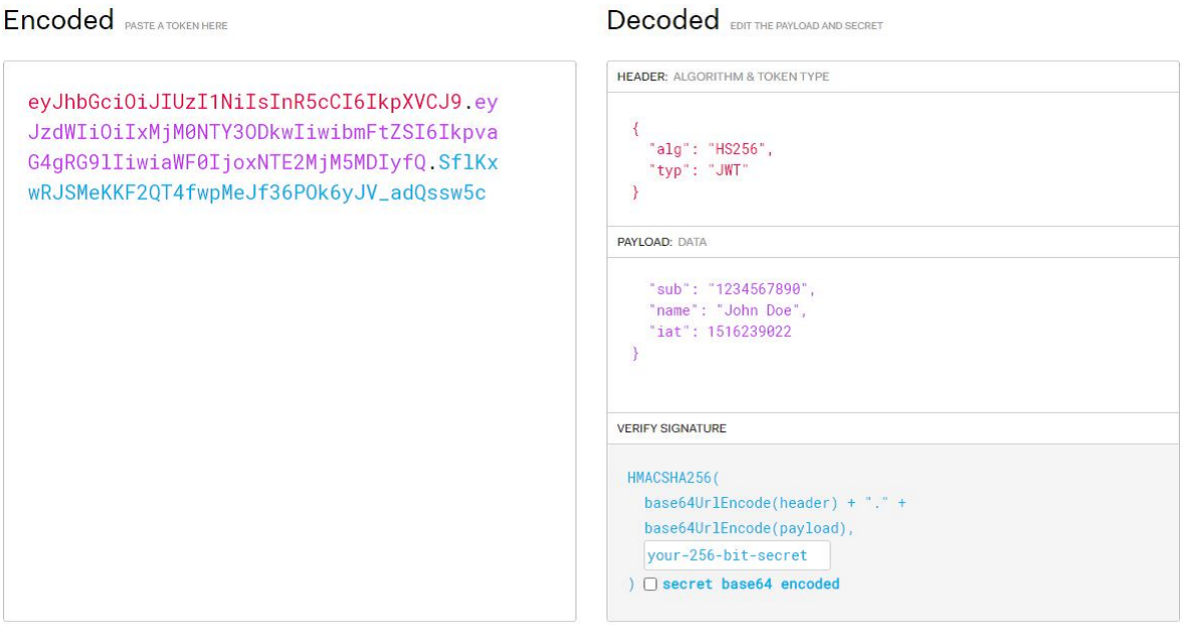


図 8　JWT の例。Encoded でユーザ情報が符号化され暗号になっているのが確認できる。Decoded で復号され Payload でユーザ情報が確認できる。

4.2 システムの詳細

本研究の開発システムは、2 つの WebAPI によって動作する。ログイン・JWT 発行 API、ユーザ認証 API である。ログイン・JWT 発行 API はユーザ ID とパスワードが既存のユーザ情報と一致するかを調べ、一致した場合 JWT を発行しユーザに返す。ユーザ認証 API はユーザが聞きたい限定ポッドキャストを選択したときにＪＷＴの検証を行った後に、正しかった場合にそのユーザがサブスクライブに登録しているのを確認し、ユーザにメディアファイルを返す。本システムのフローチャートを**図 9** に示す。

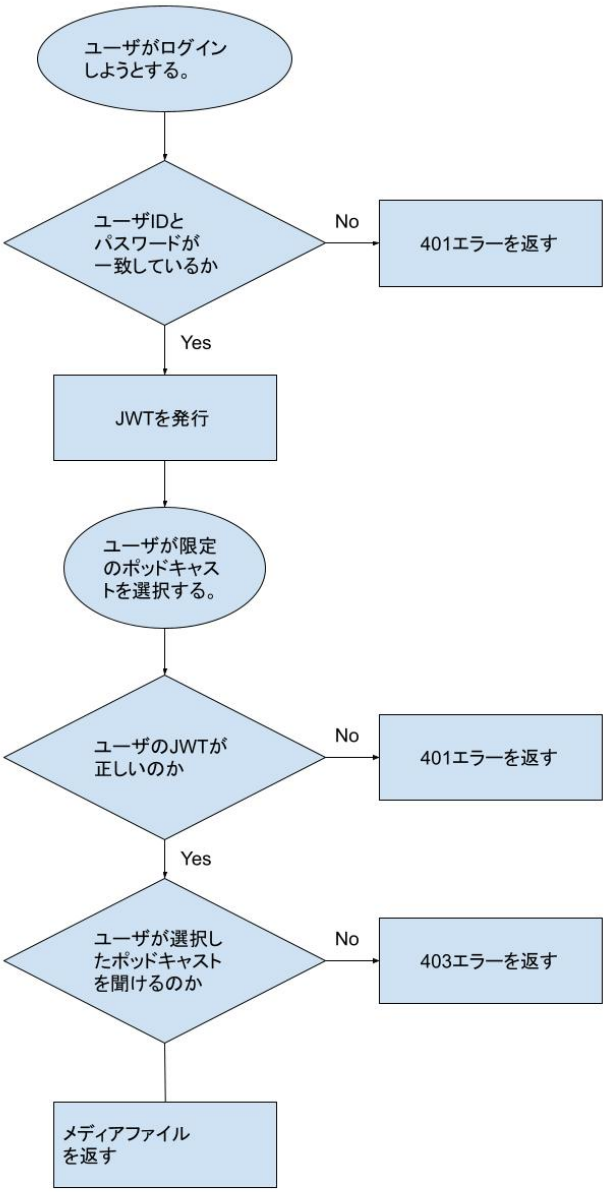


図 9　本システムのフローチャート図

4.2.1 ログイン・JWT 発行 API

ユーザがログイン画面でユーザ ID とパスワードを入力したときに、送られてきた二つの情報があらかじめ作成したユーザ情報と一致しているかの判定を行う。もし一致しない場合は HTTP エラーコード「401：Unauthorized」を返す。この場合の動作を**図 10** に示す。一致した場合は、受け取ったユーザ ID と秘密鍵を符号化し JWT を発行する。そして発行したトークンをユーザへ返す。この場合の動作を**図 11** に示す。

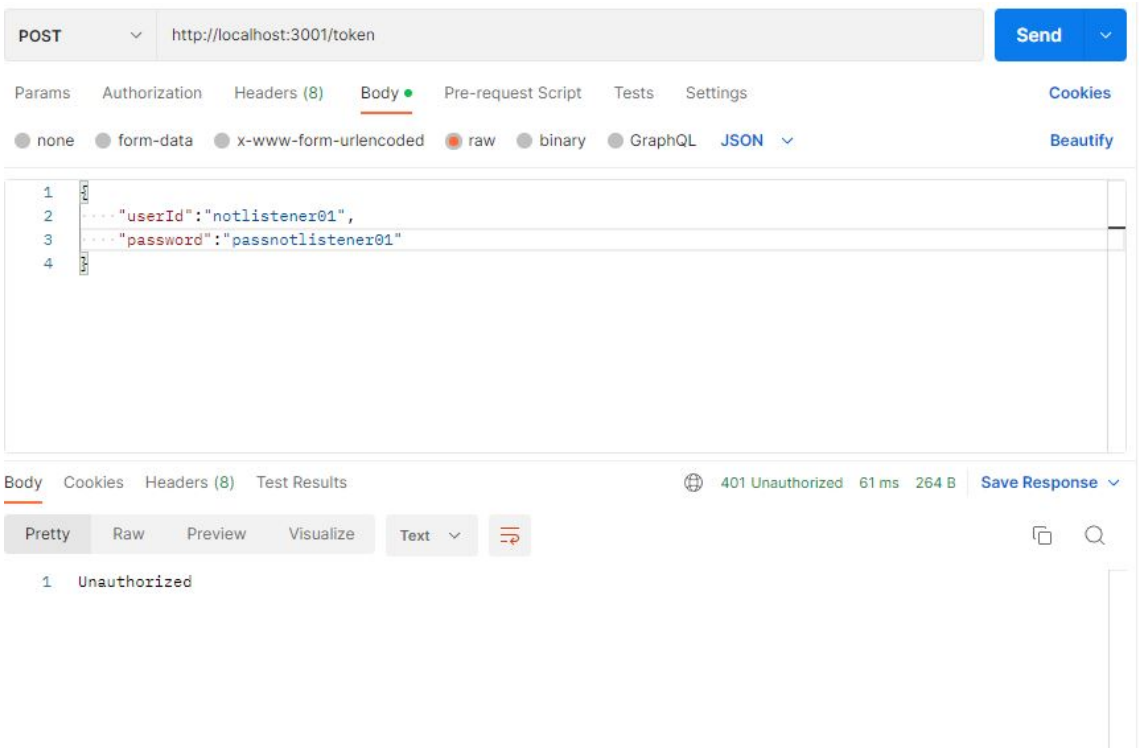


図 10　本システムにてログインに失敗した際の動作
画面上部にて本システム上に存在しないユーザ情報を JSON 形式で入力し送信した。本システム上に存在しているユーザ情報と一致せず HTTP エラーコード「401：Unauthorized」が返信され画面下部にて表示されている。

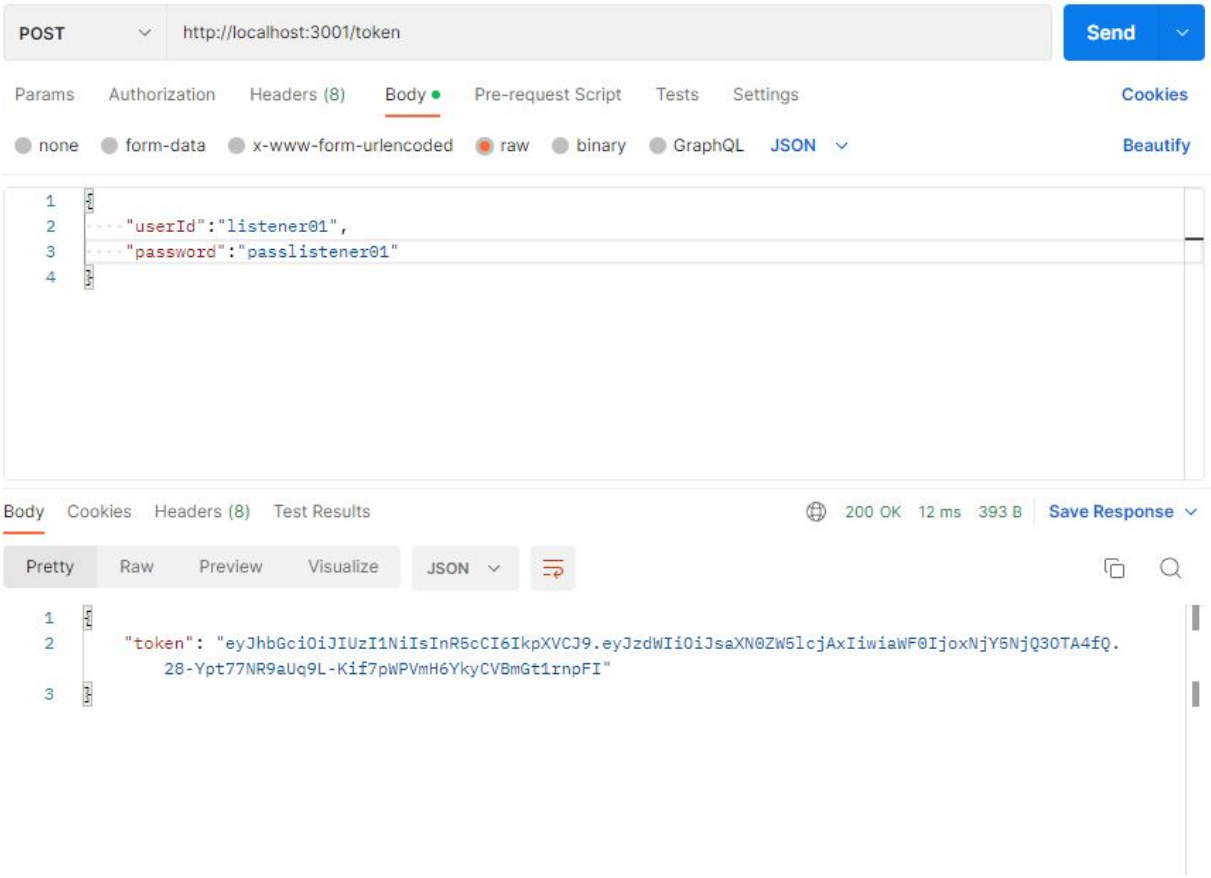


図 11　本システムにてログインに成功し JWT が発行された際の動作
画面上部にて本システム上に存在しているユーザ情報を JSON 形式で入力し送信した。本システム上に存在しているユーザ情報と一致したため JWT が発行し返信され画面下部にて表示されている。

4.2.2 ユーザ認証 API

ユーザがサブスクリプションしているユーザ限定のポッドキャストを再生しようとしたとき、ユーザ認証 API が動作する。まずは送られてきたユーザの JWT を復号化する。そのユーザ情報を確認しシステム上に存在しているユーザ情報と一致しているのかの判定を行う。もし一致しない場合は HTTP エラーコード「401：Unauthorized」を返す。この場合の動作画面を**図 12** に示す。一致した場合はサブスクリプションにそのユーザが登録しているかの判定を行う。ポッドキャストの投稿者ごとに権限を作成しており、ユーザがその投稿者のサブスクリプションに対応した権限を持っているかで判断する。もし一致しない場合は HTTP エラーコード「403：Forbidden」を返す。この場合の動作画面を**図 13** に示す。一致した場合は選択されたポッドキャストのメディアファイルを返すことでユーザがポッドキャストの視聴が可能となる。この場合の動作画面を**図 14** に示す。

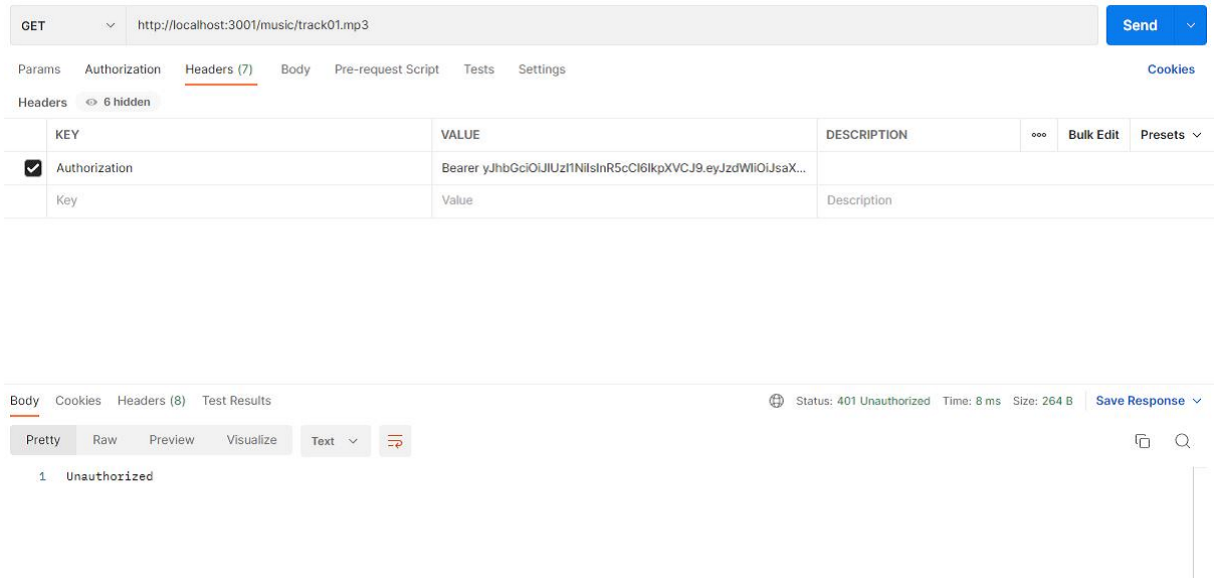


図 12　本システムにて JWT 検証で一致しない際の動作
KEY に認証用のリクエストヘッダ名である Authorization を入力し、VALUE に発行されていない改竄された JWT を入力して送信した。JWT 検証にて復号化されたユーザ情報がシステム上に存在していないため HTTP エラーコード「401：Unauthorized」が返信され画面下部にて表示されている。

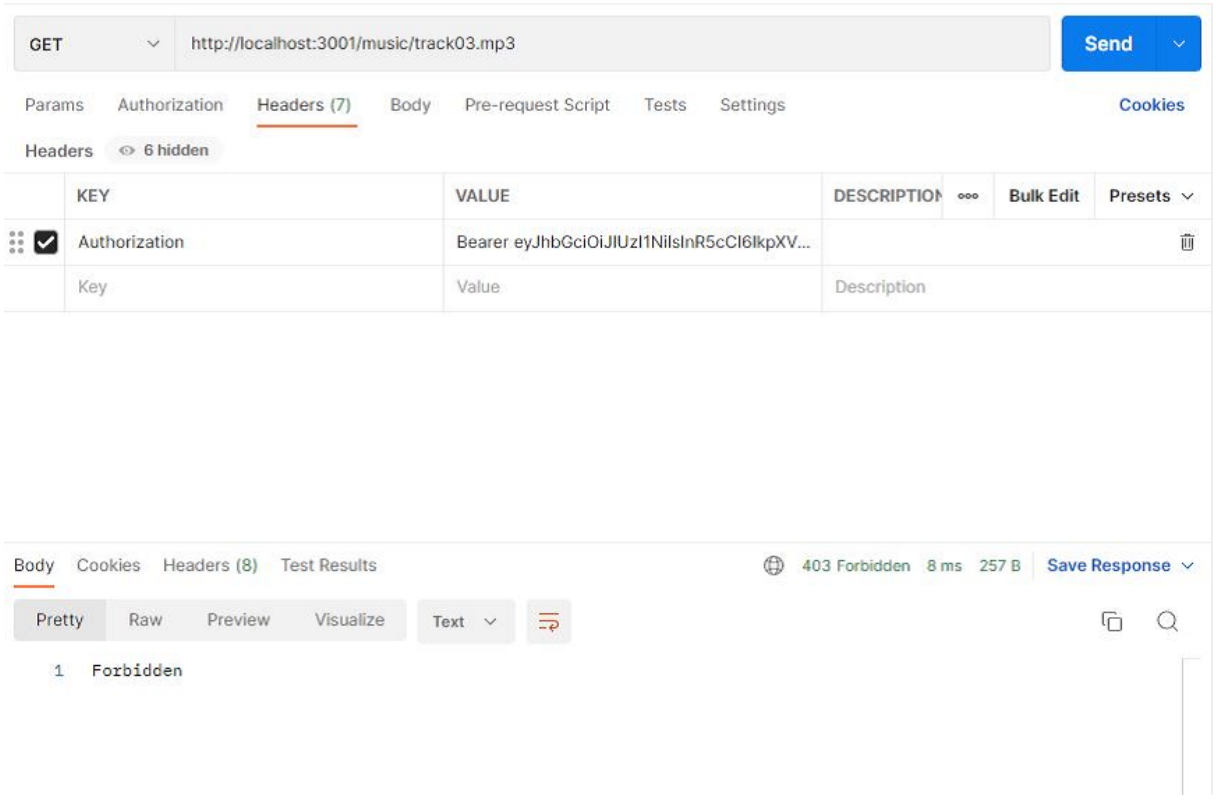


図 13　本システムにてサブスクリプションに登録していないユーザが再生しようとした際の動作
画面上部にて KEY に認証用のリクエストヘッダ名である Authorization を入力し、VALUE に別のサブスクリプションの権限を持ったユーザの JWT を入力して送信した。JWT 検証にて復号化されたユーザ情報がシステム上に存在していたが選択されたサブスクリプション限定のポッドキャストを再生する権限とは違うため HTTP エラーコード「403：Forbidden」が返信され画面下部にて表示されている。

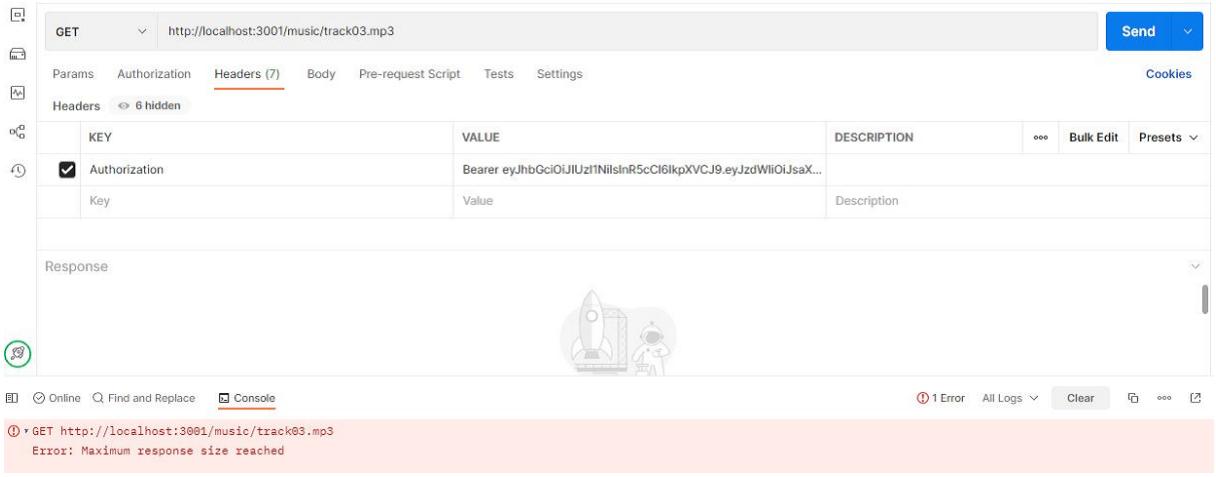


図 14　本システムにてメディアファイルを返された際の動作
画面上部にて KEY に認証用のリクエストヘッダ名である Authorization を入力し、VALUE に対応したサブスクリプションの権限を持ったユーザの JWT を入力して送信した。JWT 検証にて復号化されたユーザ情報がシステム上に存在しており、サブスクリプション限定のポッドキャストを再生する権限と一致したため、メディアファイルが返信された。大容量ファイルが返信されているのが画面下部にて表示されている。

5　検証

本システムを用いた場合に、現状のポッドキャストの抱える問題を解消できるのかの検証を行った。

5.1　再生アプリケーションでの検証

本研究にて開発したシステムの動作に対応した、ユーザ ID とパスワード、JWT の授受ができる再生アプリケーションを作成し、RSS を登録し動作の確認を行った。

1. サブスクリプションに登録したユーザでログインし、対応したサブスクリプション限定のポッドキャストを選択
2. サブスクリプションに登録したユーザでログインし、対応していないサブスクリプション限定のポッドキャストを選択
3. ログインせずに、サブスクリプション限定のポッドキャストを選択

5.2　検証結果

5.2.1　サブスクリプションに登録したユーザでログインし、対応したサブスクリプション限定ポッドキャストを選択

対応したユーザ情報でログインしたため、認証され想定通り再生することができた。その際の画面を**図 15** に示す。

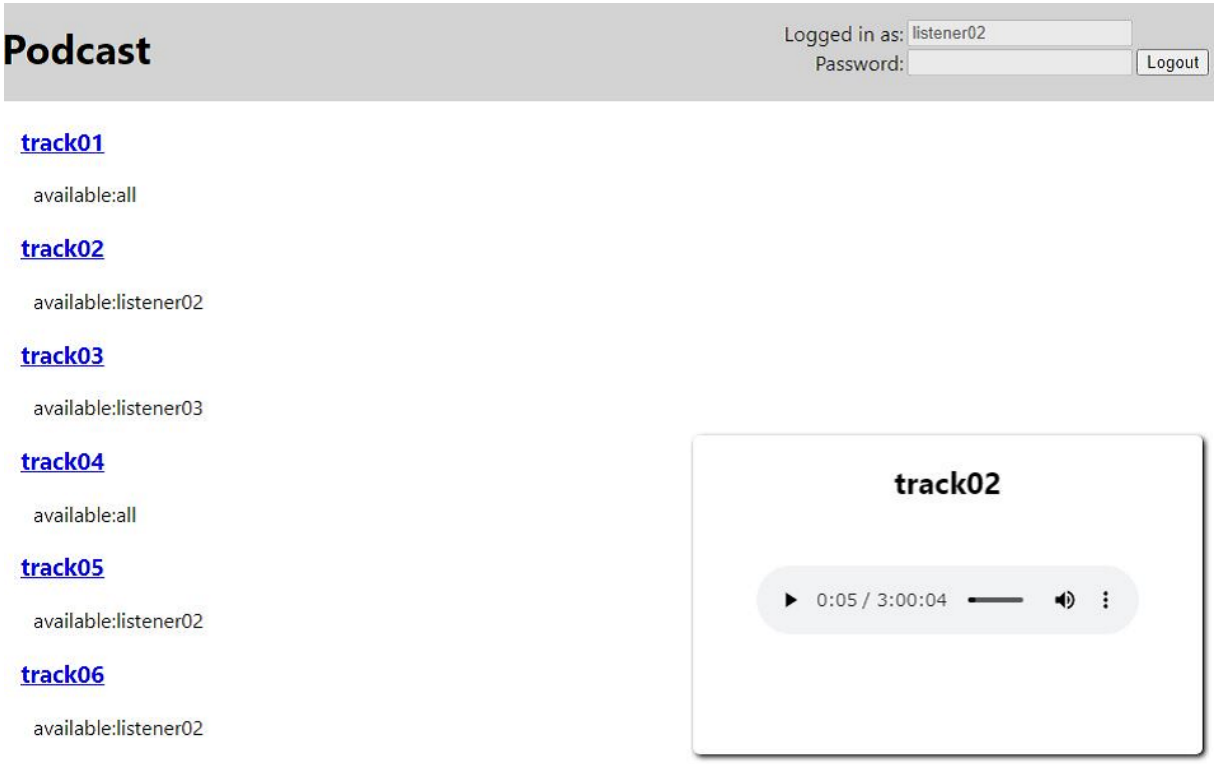


図 15　サブスクリプション限定ポッドキャスト再生に成功した際の画面
ユーザ認証画面右上のログインフォームにて listener02 がログインしている。画面左には RSS から取得したポッドキャストを列挙しておりポッドキャスト名とそのポッドキャストを聞く事のできる権限を持ったユーザが表示されている。track02 を選択し、ユーザ情報が認証されたため画面右下のメディアプレイヤーにて再生アプリケーション上での再生に成功したことが確認できる。

5.2.2 別のサブスクリプションに登録したユーザでログインし、対応していないサブスクリプション限定のポッドキャストを選択

対応したユーザ情報でログインしていないため、認証されず想定通り再生することができなかった。その際の画面を**図 16** に示す。

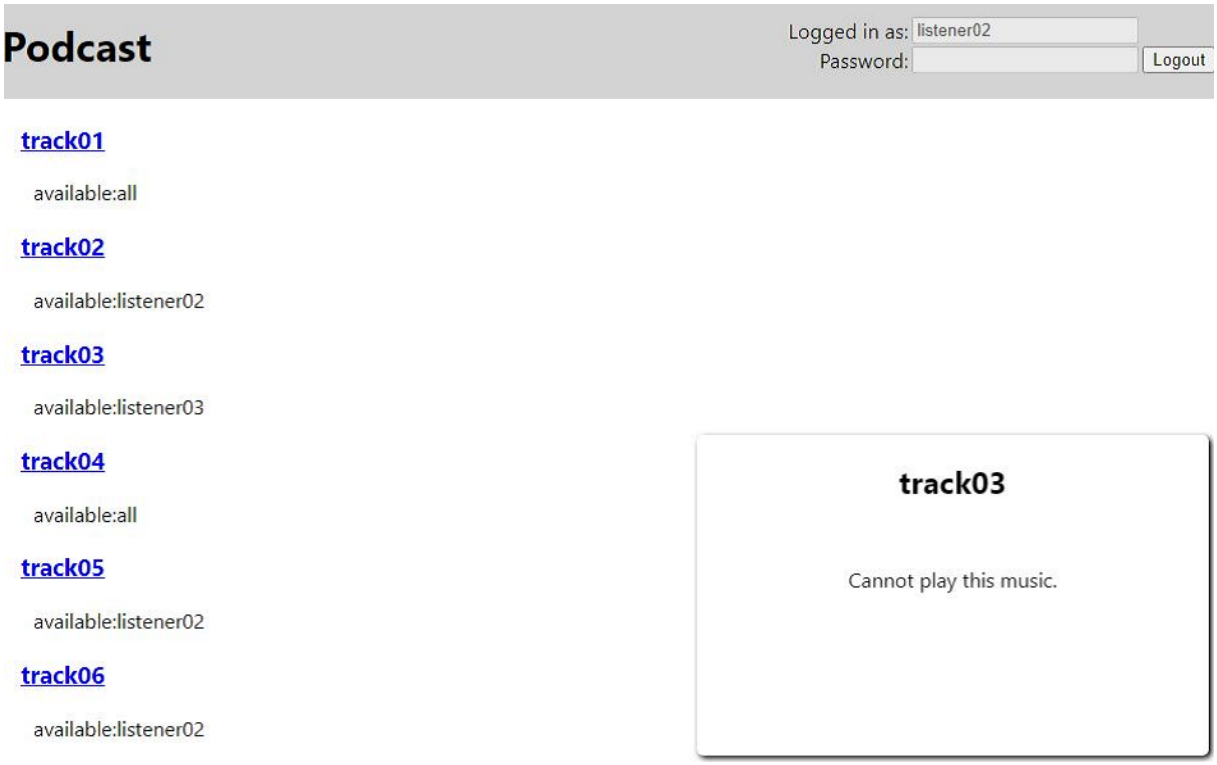


図 16　サブスクリプションに登録しているユーザが登録していないサブスクリプション限定のポッドキャストを選択し、再生に失敗した際の画面
画面右上のログインフォームにて listener02 がログインしている。画面左には RSS から取得したポッドキャストを列挙しておりポッドキャスト名とそのポッドキャストを聞く事のできる権限を持ったユーザが表示されている。track03 を選択し、ユーザ情報が認証されなかったため画面右下のメディアプレイヤーにて再生アプリケーション上での再生に失敗したことが確認できる。

5.2.3　ログインせずに、サブスクリプション限定されたポッドキャストを選択

ログインをしていないため、想定通り再生することができなかった。その際の画面を**図 17** に示す。

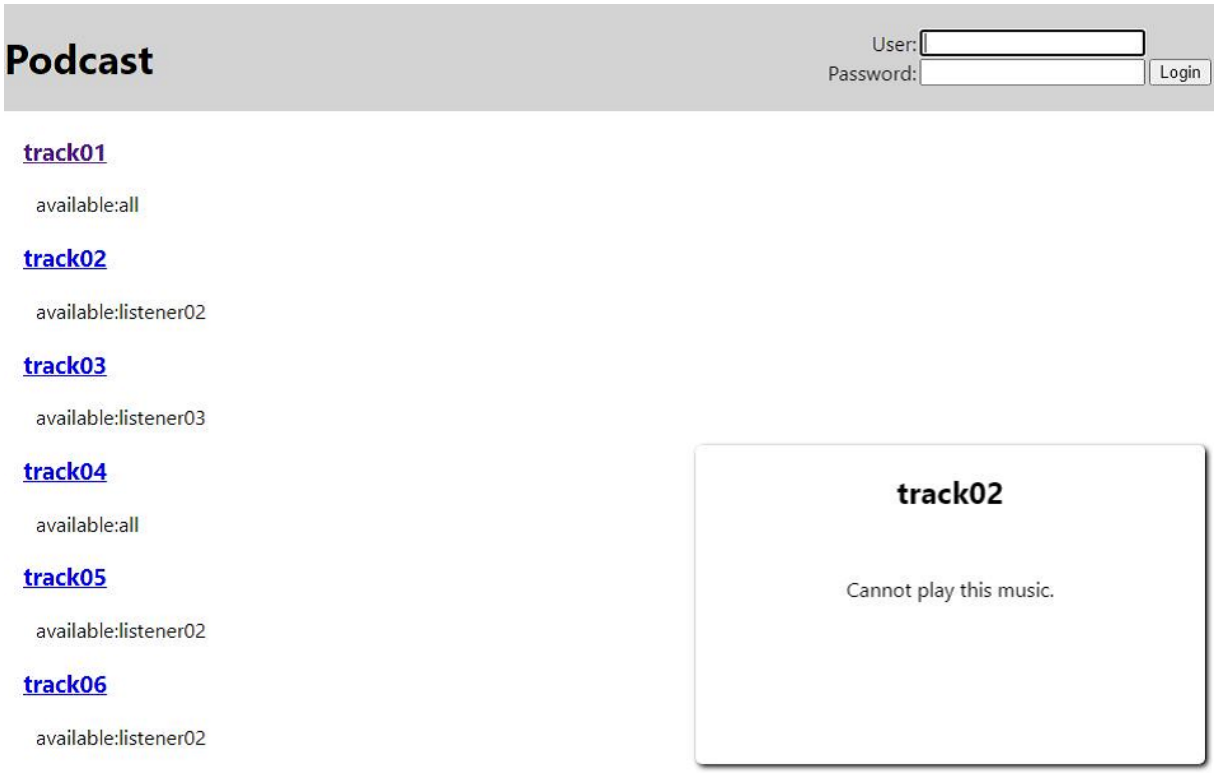


図 17　ログインをせずにサブスクリプション限定のポッドキャストを選択し、再生に失敗した際の画面
画面右上のログインフォームにてログインを行わない。画面左には RSS から取得したポッドキャストを列挙しておりポッドキャスト名とそのポッドキャストを聞く事のできる権限を持ったユーザが表示されている。track02 を選択し、ユーザ情報が認証されないため画面右下のメディアプレイヤーにて再生アプリケーション上での再生に失敗したことが確認できる。

5.3 RSS に記述されている URL からのアクセス検証

Substack のサブスクリプションサービスはユーザごとに個別の RSS を生成していたが、メディアファイル自体に認証がないため URL が流出するとだれでもアクセスすることができた。よって本システムでは、RSS のメディアファイル URL からアクセスを行うことで Substack の抱える問題を解決ができるかの検証を行った。

5.4 検証結果

作成した RSS を表示し、サブスクリプション限定のコンテンツの URL にアクセスした。アクセスは禁止され HTTP エラーコード「403：Forbidden」が表示された。作成した RSS を**図 18** に示す。アクセス後のエラーコードが表示された画面を**図 19** に示す。

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
    xmlns:itunes="http://www.itunes.com/dtds/podcast-1.0.dtd">
  <channel>
    <item>
      <title>track01</title>
      <description>available:all</description>
      <enclosure url="http://localhost:3001/music/track01.mp3"></enclosure>
    </item>
    <item>
      <title>track02</title>
      <description>available:listener02</description>
      <enclosure url="http://localhost:3001/music/track02.mp3"></enclosure>
    </item>
    <item>
      <title>track03</title>
      <description>available:listener03</description>
      <enclosure url="http://localhost:3001/music/track03.mp3"></enclosure>
    </item>
    <item>
      <title>track04</title>
      <description>available:all</description>
      <enclosure url="http://localhost:3001/music/track01_noneSE.wav"></enclosure>
    </item>
    <item>
      <title>track05</title>
      <description>available:listener02</description>
      <enclosure url="http://localhost:3001/music/track02_noneSE.wav"></enclosure>
    </item>
    <item>
      <title>track06</title>
      <description>available:listener02</description>
      <enclosure url="http://localhost:3001/music/track03_noneSE.wav"></enclosure>
    </item>
  </channel>
</rss>
```

図 18　作成した RSS をブラウザ上で表示した画面。item 要素の階層にある title 要素にタイトルを記述、description 要素にアクセスできるユーザを記述、enclosure 要素にはメディアファイルの URL を記述している。

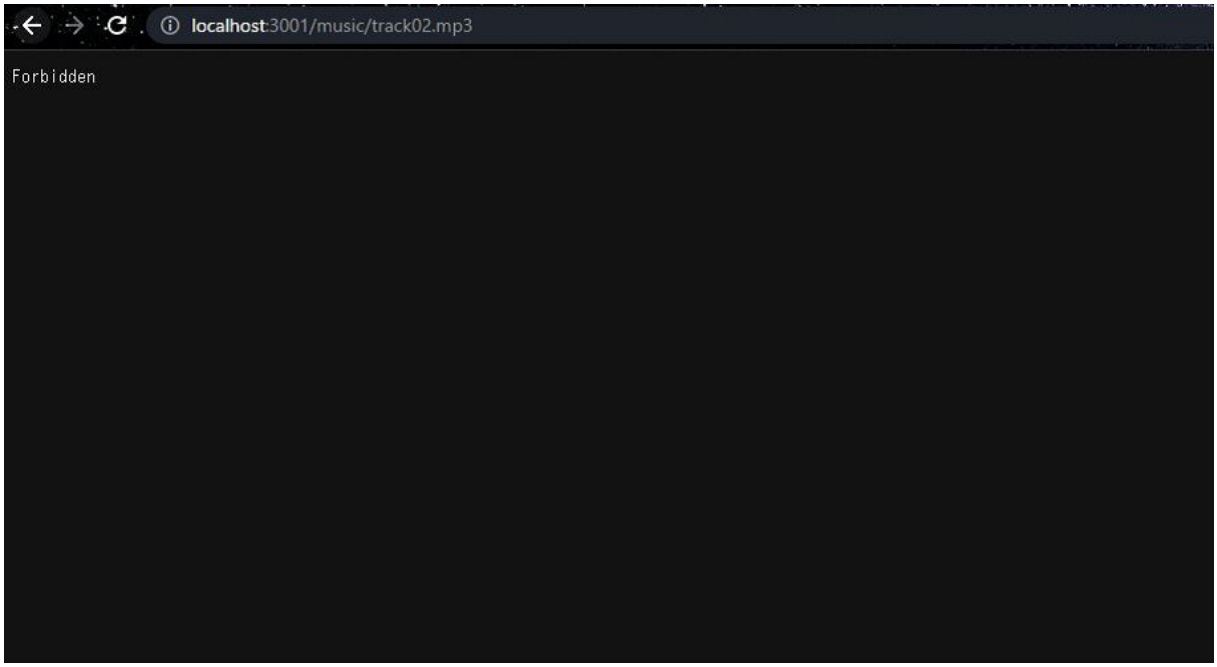


図 19　RSS のリンクからアクセスした際のブラウザ画面。認証情報を用いずアクセスしたため閲覧禁止を意味する HTTP エラーコード「403：Forbidden」が表示されている。

6 考察

本研究で開発したポッドキャストのサブスクリプションシステムは RSS を用いてユーザ認証を必要とするポッドキャストの配信を行うための手法を提案した。事項で本研究で開発したシステムの利点と問題点の考察を示す。

6.1 利点

6.1.1 RSS を用いた配信

RSS を用いてポッドキャストの更新情報の配信を行っているため、各々のユーザが各自で利用しているポッドキャストの再生アプリケーションで更新情報を受け取る事ができる。

6.1.2 ユーザ認証

メディアファイルが保存されているサーバに認証を実装したため、RSS に記述されているメディアファイルの URL にアクセスしてもアクセスが禁止される。よってサブスクリプション限定のメディアファイル URL が流出し、誰でもアクセスリクエストをすることが可能になってもアクセスを拒否する事ができる。

6.2 課題点

6.2.1 既存のポッドキャスト再生アプリケーションで視聴ができない

既存のポッドキャスト再生アプリケーションは本システムの認証情報を授受するための機能が実装されていないため、本システムのサブスクリプション限定コンテンツの再生ができない。

6.2.2 ユーザ側から新しくアカウント登録することができない

本システムはあらかじめ作成したアカウントしか利用する事ができず、システム管理者しかユーザを追加することができない。新規ユーザを受動的に増やすことができないのでサービスを展開するにあたってマネタイズを行いづらい仕組みになっている。よってユーザを自動的に登録するシステムを開発する必要がある。

6.2.3 ユーザ側から新しくサブスクリプションを登録することができない

本システムはあらかじめ作成したアカウントにサブスクリプションを利用できるように権限を与えているので、システム管理者しかユーザに権限を付与することができない。よってこれらを自動的に登録できるシステムを開発する必要がある。

7　結論

本研究では JWT 認証を用いたをポッドキャストサブスクリプションサービスのシステムの開発を行った。その結果ポッドキャストの標準方式を満たし、JWT 認証を用いたポッドキャストサブスクリプションサービスを開発することができた。しかし、既存のポッドキャスト再生アプリケーションでは本システムの認証に必要なユーザ情報・JWT の授受をするための機能が実装されていないため本システムを利用することができなかった。各サービスプロバイダが認証情報を授受するための機能を追加され、標準化が進むことで本システムの利用が可能になると考える。

謝辞

本研究の研究作業を進めていく上で大垣斉准教授からご協力及びご指導頂き、深く感謝致します。また、情報教育システム研究室のメンバー及び卒業生の方々に深く感謝の意を表します。

参考文献

[1] Podcast listeners worldwide*, 2019-2024 (millions, % change, and % of internet users) — insider intelligence. https://www.insiderintelligence.com/chart/251021/podcast-listeners-worldwide-2019-2024-millions-change-of-internet-users, July 2021. (Accessed on 11/17/2022).

[2] ポッドキャスト国内利用実態調査 2020 — 株式会社オトナル、株式会社朝日新聞社. https://otonal.co.jp/pdf/podcast-report-in-japan2020.pdf, January 2021. (Accessed on 11/17/2022).

[3] Submit a show - apple podcasts for creators. https://podcasters.apple.com/support/897-submit-a-show. (Accessed on 12/22/2022).

[4] Substack for podcasts. https://substack.com/podcasts?utm_source=menu-dropdown. (Accessed on 12/22/2022).

[5] Jwt (json web token) とは - 意味をわかりやすく - it 用語辞典 e-words. https://e-words.jp/w/JWT.html. (Accessed on 11/28/2022).

付録 A　WebAPI のソースコード

A.1　ソースコード.1

Listing 1　app.js

```
import express from 'express';
import jwt from 'jsonwebtoken';
import cors from 'cors';
import { dirname } from 'path';
import { fileURLToPath } from 'url';
import { MusicService } from './service/MusicService.js';
import { UserRepository } from './repository/UserRepository.js';

const __dirname = dirname(fileURLToPath(import.meta.url));
const jwtSecret = 'secret';

const app = express();

app.use(cors({origin: true}));
app.use(express.json());

// Authentication
app.use((req, res, next) => {
    try {
        const jwtToken = req.get('Authorization')?.substring(7);
        if (jwtToken) {
            const decoded = jwt.verify(jwtToken, jwtSecret);
            const userId = decoded.sub;
            const ur = new UserRepository();
            const user = ur.getByUserId(userId);
            if (user) {
                req.user = user;
            }
        }
        next();
    } catch (err) {
        res.sendStatus(401);
    }
});
```

```
app.use('/music/:name', (req, res, next) => {
    const ms = new MusicService();
    if (ms.isMusicAvailableFor(req.params.name, req?.user?.id)) {
        next();
    } else {
        res.sendStatus(403);
    }
});

app.use('/music', express.static(__dirname + '/../static'));

// Login & Issuance
app.post('/token', (req, res) => {
    const userId = req.body?.userId;
    const password = req.body?.password;
    let valid = false;
    if (userId) {
        const ur = new UserRepository();
        const user = ur.getByUserId(userId);
        if (user && user.password === password) {
            const jwtToken = jwt.sign({ sub: userId }, jwtSecret);
            res.json({ token: jwtToken });
            valid = true;
        }
    }
    if (!valid) {
        res.sendStatus(401);
    }
});

app.get("/", (req, res) => {
    res.sendFile('./rssFeed.rss', {
        root: __dirname,
    });
});

const server = app.listen(3001, () => {
    console.log('server started');
});
```

A.2 ソースコード.2

Listing 2 MusicService.js

```
import { MusicRepository } from '../repository/MusicRepository.js';
import { SubscriptionRepository } from '../repository/SubscriptionRepository.js';

export class MusicService {
    isMusicAvailableFor(musicName, userId) {
        const mr = new MusicRepository();
        const music = mr.getByName(musicName);

        const sr = new SubscriptionRepository();
        const subscription = sr.getByUserId(userId);

        return music?.isPublic || subscription?.subscribingUserIds?.includes(music.owner);
    }
}
```

A.3 ソースコード.3

Listing 3 MusicRepository.js

```
import { Music } from '../model/Music.js';

export class MusicRepository {
    getAll() {
        return [
            new Music('track01.mp3', 'user01', true),
            new Music('track02.mp3', 'user01', false),
            new Music('track03.mp3', 'user02', false),
            new Music('track01_noneSE.wav', 'user03', true),
            new Music('track02_noneSE.wav', 'user03', false),
            new Music('track03_noneSE.wav', 'user03', false),
        ];
    }

    getByName(name) {
        return this.getAll().find(music => music.name === name);
    }
}
```

A.4 ソースコード.4

Listing 4 SubscriptionRepository.js

```
import { Subscription } from '../model/Subscription.js';

export class SubscriptionRepository {
    getByUserId(userId) {
        const subscriptions = [
            new Subscription('listener01', []),
            new Subscription('listener02', ['user01', 'user03']),
            new Subscription('listener03', ['user02'])
        ];

        return subscriptions.find(subscription => subscription.userId === userId);
    }
}
```

A.5 ソースコード.5

Listing 5 UserRepository.js

```
import { User } from '../model/User.js';

export class UserRepository {
    getByUserId(userId) {
        const users = [
            new User('user01', 'passuser01'),
            new User('user02', 'passuser02'),
            new User('user03', 'passuser03'),
            new User('listener01', 'passlistener01'),
            new User('listener02', 'passlistener02'),
            new User('listener03', 'passlistener03'),
        ];

        return users.find(user => user.id === userId);
    }
}
```

A.6 ソースコード.6

Listing 6 Music.js

```
export class Music {
    constructor(name, owner, isPublic) {
        this._name = name;
        this._owner = owner;
```

```
        this._isPublic = isPublic;
    }

    get name() {
        return this._name;
    }

    get owner() {
        return this._owner;
    }

    get isPublic() {
        return this._isPublic;
    }
}
```

A.7 ソースコード.7

Listing 7 Subscription.js

```
export class Subscription {
    constructor(userId, subscribingdUserIds) {
        this._userId = userId;
        this._subscribingUserIds = subscribingdUserIds;
    }

    get userId() {
        return this._userId;
    }

    get subscribingUserIds() {
        return this._subscribingUserIds;
    }
}
```

A.8 ソースコード.8

Listing 8 User.js

```
export class User {
    constructor(id, password) {
        this._id = id;
        this._password = password;
    }
```

```
    get id() {
        return this._id;
    }

    get password() {
        return this._password;
    }
}
```

付録 B　再生アプリケーションのソースコード

B.1　ソースコード.1

Listing 9　securityContextHolder.js

```
class SecurityContextHolder {
    constructor() {
        this._isLoggedIn = false;
        this._userId = null;
        this._jwtToken = null;
    }

    getContext() {
        if (this._isLoggedIn) {
            return {
                userId: this._userId,
                jwtToken: this._jwtToken
            };
        } else {
            return null;
        }
    }

    setContext(userId, jwtToken) {
        this._isLoggedIn = true;
        this._userId = userId;
        this._jwtToken = jwtToken;
    }

    clearContext() {
        this._isLoggedIn = false;
```

```
    }
}

export default new SecurityContextHolder();
```

B.2 ソースコード.2

Listing 10 requestBackend.js

```
import { backendDomain } from './config.js';
import securityContextHolder from './securityContextHolder.js';

function requestBackend(pass, options) {
    const context = securityContextHolder.getContext();
    if (context) {
        options = options || {headers: {}};
        options.headers['Authorization'] = 'Bearer ' + context.jwtToken;
    }
    return fetch(backendDomain + pass, options);
}

export default requestBackend;
```