Node.js Web アプリケーションにおける高負荷耐性の実証的分析と改善手法 —会津地域バスロケーションシステムの実運用を目指して—

五十嵐 妃菜

1. はじめに

公共交通や観光情報を扱うWebアプリケーション（以下Webアプリ）は，利用者への情報提供や行動支援の手段として広く利用されている．これらのシステムでは，イベント時や観光シーズン等に多数の利用者が同時にアクセスする状況が発生しやすく，高負荷時でも安定して動作することが求められる．

実運用環境を想定したWebアプリでは，アクセス集中により応答時間の増大等の問題が生じ，利用者の利便性やサービスの信頼性を低下する恐れがある．しかしながら，このような条件下で，ボトルネック要因と改善効果を定量的に評価した検証は十分に行われていない．

そこで本研究では，多数の同時アクセスを想定して利用されるNode.jsを用いたWebアプリを対象とし，負荷試験によって高負荷時の性能低下要因を特定し，改善手法の有効性を定量的に評価する．

2. Node.js Web アプリケーションの高負荷耐性に関する背景と課題

2.1 Node.js の仕組みと特徴

Node.jsは，サーバ側でJavaScriptを実行する環境であり，WebアプリやWeb APIの実装に広く利用されている[1]．多数の同時アクセスを扱うことを想定して設計されているため，I/Oの処理に伴う待機で他の処理が停滞しにくい点が特徴である．

Node.jsの中心となる仕組みは，非同期I/Oモデル[1]とイベントループである[1]．Node.jsは，JavaScriptの実行自体は基本的に単一スレッドである一方，入出力処理は非同期に開始され，イベントループが完了通知に基づきコールバック等を順次実行する．これにより，入出力待機によるブロッキングを抑えつつ多数のリクエストを効率的に処理できる利点がある[2]．

一方で，単一スレッドで実行するという制約のため，CPU資源[2]を集中的に消費する計算が長く続くと，イベントループが占有されやすい[3]．その結果，応答時間の増大，タイムアウト，エラー発生率の増大といった性能上の問題が生じる可能性がある．この問題は，多数のリクエストが同時に発生する高負荷状況において特に顕在化しやすい[3]．これらの挙動を定量的に評価するためには，応答時間，エラー発生率，CPU使用率，メモリ使用量，およびイベントループ遅延等の指標を用いることが重要である[4]．本研究においても，これらの指標に基づき，高負荷状態におけるシステム挙動を評価し，ボトルネックとなる要因を明らかにする．

2.2 高負荷耐性検証の対象システム

本研究では，実運用時に想定されるアクセス集中下での挙動を評価するため，桐生[5]，鈴木[6]，黒須[7]，安田[8]が開発・改良したバスロケーションシステムを対象とする．

桐生[5]はバスの待ち時間を活用し，ユーザの空腹度や疲労度を考慮した観光スポットの推薦システムを開発した．また，鈴木[6]はバス停の位置やバスの現在位置を地図上にリアルタイムで表示するシステムを開発した．これら2つのシステムを統合することで，地図描画，位置情報の取得，およびリアルタイム更新処理を含むWebアプリとしてのバスロケーションシステムが構築された．さらに，黒須[7]および安田[8]は同システムを基に，ユーザビリティと応答性の向上を目的とした改良を行った．具体的には，マーカークラスタリングの導入による地図描画時の情報量削減や，表示範囲に応じたバス停データ取得手法が提案されている．これらの改良は，地図描画に伴うデータ量やデータ取得回数の増大といった，高負荷状態に顕在化しやすい問題への対処を意識したものである．

このように，当該バスロケーションシステムは，実運用を想定したWebアプリであり，不特定多数の利用者による同時アクセスが発生し得る構成となっている．また，地図描画に伴う大量データの処理，外部APIとの連携，データベース参照，およびリアルタイムでの情報更新を含むため，高負荷状態では複数処理が同時にボトルネックとなる可能性を有している．

しかし，これらの研究では，アクセス集中による高負荷状態におけるシステム挙動の評価は十分に行われていない．既存システムはNode.jsを用いて構築されたWebアプリであり，実運用環境におけるアクセス集中時の挙動を通して，Node.jsの高負荷耐性を検証する対象として適している．

そこで本研究では，本バスロケーションシステムを対象にアクセス集中時の挙動を測定し，パフォーマンス低下要因の特定と改善手法の検討を行うことを目的とする．

3. 検証方法および実験環境

3.1 検証対象システムの構成

対象システムは，サーバ側・クライアント側・外部データの三つの構成要素からなる．サーバ側では，Node.js上でExpressを用いたWebサーバが稼働し，リクエストに応じてデータ抽出，整形，JSON化，および圧縮等を行った上で，HTTPSを用いてクライアントへ応答する．外部データの取得やレスポンス送信は非同期I/Oモデルにより実行される一方，データ抽出，整形，JSON化，および圧縮といった処理は同期的に実行される．これらの計算処理が高負荷状態において長時間実行されると，イベントループ遅延が増大して応答遅延やCPU負荷増大の要因となる可能性がある．

クライアント側では，Google Maps API[3]を用いて地図表示を行う．ユーザ操作や初期画面表示要求に応じてサーバへデータリクエストを送信し，返却されたJSONデータに基づき，地図上にバス停情報やバスの位置情報を動的に表示する．

外部データでは，公共交通データのGTFS（General Transit Feed Specification）データ[4]を使用する．GTFS-JPに基づく静

---

[1] 入出力処理の完了を待たずに次の処理を進める方式

[2] CPU コア上で計算処理を実行するための処理能力，およびプロセスに割り当てられる CPU 時間

[3] https://developers.google.com/maps?hl=ja

[4] https://www.gtfs.jp/get-started.html

的データとしてバス停情報および路線情報を取得するとともに，GTFSリアルタイムに基づく動的データとしてバスの位置情報および遅延時間を取得する．サーバ側では，静的データと動的データを統合して処理し，クライアントに送信する．

以上の構成により，クライアントからの要求に応じてバス停情報・路線情報・バスの位置や遅延情報がJSONで返され，クライアント側においてGoogle Maps上へ動的に表示される．

3.2 実験環境

本研究では，実運用を想定したクラウド型サーバ構成を再現するため，GMOインターネットが提供するConoHa Windows Server[5]（Microsoft Windows Server 2025 Datacenter）を採用した．使用したサーバ環境は，Intel Xeon Ice Lake世代のプロセッサを搭載し，多数のCPUコアおよび十分なメモリ帯域を有している．また，ネットワーク帯域は上り・下りともに最大100Mbpsの環境である．

アクセス集中時におけるパフォーマンス低下の要因を切り分けるため，CPUおよびメモリ構成の異なる3段階（低スペック，中スペック，高スペック）のサーバ環境を用意し，CPU使用率，メモリ使用量，およびイベントループ遅延を含む指標を併用して挙動を比較・検証した．検証に用いた各サーバスペックは以下の通りである．

- 低スペック:メモリ　8GB　/ CPU　6Core
- 中スペック:メモリ 16GB　/ CPU　8Core
- 高スペック:メモリ 32GB　/ CPU　12Core

また，Node.js 20.17.0を用いてシステムを実行し，Node.jsアプリケーションの実行時パフォーマンスを診断するツールであるClinic.js Doctor[6]を利用して，CPU使用率，メモリ使用量，およびイベントループ遅延を計測した．

3.3 負荷試験ツールの概要と高負荷条件の設定

本研究では，Webアプリに対する高負荷状態を再現するため，負荷試験ツールとしてk6[7]を使用した．k6は，JavaScriptにより負荷条件を記述でき，同時アクセス数や負荷継続時間を柔軟に制御可能である．また，コマンドライン操作によって実行可能であり，負荷試験結果をCSV形式で出力できるため，集計や比較が容易で実験結果の分析に適している．

負荷試験では，同時アクセス数を10刻みで10から100まで[8]変化させて実施した．本研究の負荷試験では，ブラウザ操作を再現するのではなく，操作に伴ってクライアントからサーバへ送信されるHTTPSリクエストを模擬し，その送信順序と回数を負荷条件とした．リクエストタイムアウトは，実運用環境を想定し60秒に設定した．実行時間は5分間とし，安定した結果が得られるサンプル数を確保した．

4. 既存システムの負荷試験と課題分析

4.1 既存システムにおける初期表示負荷試験

3.3節で設定した負荷条件に基づき，既存システムに対して初期表示負荷試験を実施した．初期表示はすべての利用者が必ず実行する共通操作であり，同時アクセス時に負荷が集中しやすい処理であるため，本試験の対象とした．評価は表1に示す指標に基づいて行い，以下の手順で実施した．

1. バスロケーションシステムにアクセス
2. HTTPステータスコードが400未満であり，タイムアウトが発生していないことを確認
3. 上記操作を繰り返し実行する

表 1に既存システムでの初期表示負荷試験の結果を示す．紙面の関係で，最大負荷時（同時アクセス数100）の結果のみを示す．

表 1 「既存システムでの初期表示」結果

評価項目	8GB	16GB	32GB
エラー発生率(%)	72.73	66.84	73.02
平均応答時間(s)	59.73	59.46	53.25
最大応答時間(s)	60.00	60.00	60.00
CPU使用率(%)	0.11	0.14	0.10
メモリ使用量(MB)	140.75	137.86	132.67
平均イベントループ遅延(s)	0.06	0.06	0.06

4.2 既存システムの課題

表 1に示す結果から，サーバリソースには余裕がある一方で，応答遅延とエラーの頻発といった課題が明らかになった．

Node.jsプロセスのCPU使用率は約0.1%，メモリ使用量は140MB程度と低く，サーバリソースが逼迫していないことが確認された．しかし，エラー発生率は約70%と高く，最大応答時間がいずれのサーバスペックにおいても60秒に達している．これは，設定したタイムアウトに至ったリクエストが多数発生していることを示している．また，平均応答時間は約53～60秒と非常に長く，ユーザ体験の観点から問題があることが確認された．加えて，3つのサーバスペックで結果に大きな差が見られないことから，性能低下の要因はCPUやメモリではなく，Node.js上で実行される同期的な処理を含むシステム実装に起因していると考えられる．特に，データ抽出，整形，JSON化，および圧縮といった処理がイベントループを占有し，リクエスト処理の滞留を招いている可能性が高い．

以上の結果から，既存システムにおいては，処理の分割や軽量化，非同期処理構成の見直しが有効な改善策である可能性が示唆された．

5. システム改善と性能評価

5.1 ファイル分割と静的化による構造改善

4.2節で示した課題を踏まえ，本研究ではサーバ側の処理の負荷軽減を目的として，ファイル分割と静的化による構造改善を実施した．既存システムでは，初期画面表示に伴うHTMLの動的生成処理がリクエストごとに実行され，アクセス集中時の応答遅延に関与している可能性があると考えられた．また，約2700行に及ぶ単一のJavaScriptファイルに，Expressサーバの初期化・設定，APIエンドポイントの定義，データの読み込み・処理，HTMLテンプレートの生成（クライアント側JavaScriptを含む）等が混在していた．その結果，初期表示リクエストでは不要なHTML生成やデータ整形処理が実行され，サーバ側の処理負荷が増大する構造となっていた．

そこで，本研究では，サーバ側処理（APIエンドポイントやGTFSデータ処理）とクライアント側処理（Google Maps API操作やUI制御）を分離し，HTMLテンプレートおよびスタイルシートを静的ファイルとした．これにより，初期画面表示時にサーバ側でHTMLを動的生成する処理を不要とするとともに，イベントループ遅延の要因となる処理を削減し，応答時間の短縮と高負荷耐性の改善を図る．

表 2にファイル分割後の各ファイルの役割を示す．また図1に既存システムとファイル分割後のシステム構成を示す．この改善により，サーバ側で連続的に実行されていたCPU負荷の高い処理の削減を図り，Node.jsの非同期I/Oモデルを活か

[5] https://vps.conoha.jp/windows/

[6] https://clinicjs.org/

[7] https://k6.io/

[8] 協力先の会津バス様からの情報に基づいて，上限設定が妥当であることを確認している．

しつつ，実運用時の高負荷耐性の向上が期待される．

表 2 それぞれのファイルの役割

ファイル名	役割
config.js	バージョン情報および設定値の管理
server.js	サーバ初期化，APIエンドポイント[9]， ミドルウェア設定，HTTPS設定
app.js	Google Maps API操作，マーカー処理，UI制御
index.html	HTMLテンプレート（静的ファイル）
style.css	スタイルシート（静的ファイル）

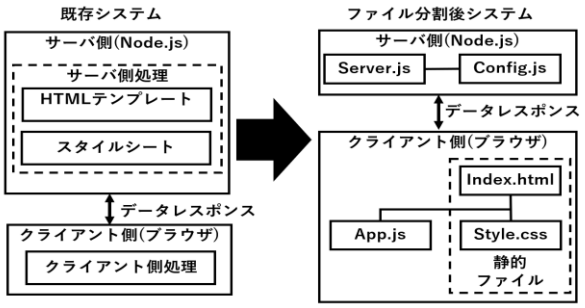

図 1 既存システムとファイル分割後システム

5.2 初期表示機能の性能評価

ファイル分割をしたシステムで負荷試験を実施した．3.3節で設定した負荷条件に基づき，同時アクセス数10から100までの範囲で試験を行った結果，すべての同時アクセス数においてエラー発生率は0%であった．そのため，最も厳しい条件である最大負荷時（同時アクセス数100）の結果のみを表 3に示す．また，ファイル分割前後の性能比較を表 4に示す．

表 3 「改善後システムでの初期表示」結果

評価項目	8GB	16GB	32GB
エラー発生率(%)	0.00	0.00	0.00
平均応答時間(s)	0.04	0.03	0.03
最大応答時間(s)	1.95	1.15	1.25
CPU 使用率(%)	0.17	0.23	0.13
メモリ使用量(MB)	111.22	109.89	111.09
平均イベントループ遅延(s)	0.00	0.00	0.00

表 4 ファイル分割前後の性能変化

評価項目	8GB	16GB	32GB
エラー発生率差(pt)	-72.73	-66.84	-73.02
平均応答時間差(s)	-59.69	-59.43	-53.22
最大応答時間差(s)	-58.05	-58.85	-58.75
CPU使用率差(%)	+0.06	+0.09	+0.03
メモリ使用量差(MB)	-29.53	-27.97	-21.58
平均イベントループ遅延差(s)	-0.06	-0.06	-0.06

ファイル分割により，エラー発生率はすべてのサーバスペックで0%となり，約70ポイントの改善が確認された．また，平均応答時間は全体で約53～60秒短縮され，メモリ使用量も25MB前後削減された．平均イベントループ遅延についても0.06秒改善された．既存システムでは，リクエストの約70%がタイムアウトにより処理未完了となり，その結果，イベントループ遅延が見かけ上高く計測されていたと考えられる．改善後システムでは，すべてのリクエストが正常に処理され，イベントループのブロッキングが解消されたことで，遅延が減少した可能性がある．また，初期表示時のHTML動的生成の廃止が，タイムアウト解消につながったと考えられる．

5.3 機能別負荷特性の分析

本節では，改善後システムの主要機能について，3.3節と同一の負荷条件で負荷試験を実施した．各試験では，機能実行に必要なデータ取得を行った後に当該機能を実行した．スポット推薦機能では，バス停データ取得後に，ランダム座標からの推薦（3回），主要5地点からの推薦（各1回），会津若松駅固定でアンケート条件を変えた推薦（4パターン）を実行した．すべてのバス表示機能では，バス停データと時刻表データ取得後にすべてのバス表示を実行した．各バスの経路表示機能では，バス停データとバスデータ取得後にランダムに選択したバスの経路表示を実行した．地図スクロール操作では，表示範囲変更を5回行った後に広範囲表示を実行した．

CPU使用率，メモリ使用量，平均イベントループ遅延は，サーバスペック間で大きな差が見られなかったため，参考値として掲載した．一方，エラー発生率および応答時間については，同時アクセス数による変動が確認されたため，図 2および図 3にすべてのバス表示機能における推移を，図 4に各バス経路表示機能における推移を示す．

表 5 「改善後システムでのスポット推薦機能」結果

評価項目	8GB	16GB	32GB
エラー発生率(%)	0.13	0.14	0.11
平均応答時間(s)	0.07	0.07	0.07
最大応答時間(s)	4.02	4.19	4.82
CPU使用率(%)	0.32	0.38	0.29
メモリ使用量(MB)	177.05	199.19	175.29
平均イベントループ遅延(s)	0.00	0.00	0.00

表 6 「改善後システムでのすべてのバス表示」結果

評価項目	8GB	16GB	32GB
エラー発生率(%)	7.50	6.43	6.25
平均応答時間(s)	6.13	6.70	6.10
最大応答時間(s)	60.00	60.00	60.00
CPU 使用率(%)	0.26	0.37	0.28
メモリ使用量(MB)	343.25	402.84	369.96
平均イベントループ遅延(s)	0.02	0.09	0.02

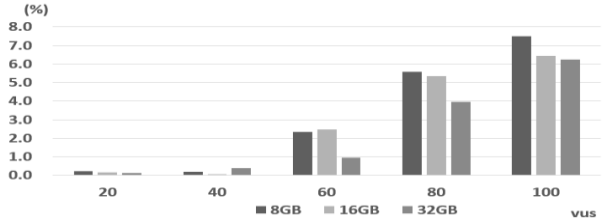

図 2 すべてのバス表示機能：エラー発生率の推移

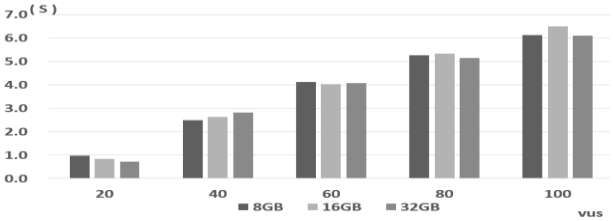

図 3 すべてのバス表示機能：平均応答時間の推移

表 7 「改善後システムでの各バスの経路表示」結果

評価項目	8GB	16GB	32GB
エラー発生率(%)	0.00	0.00	0.00
平均応答時間(s)	1.70	1.69	1.70
最大応答時間(s)	12.21	12.53	12.82
CPU 使用率(%)	0.46	0.38	0.46
メモリ使用量(MB)	342.24	262.04	320.80
平均イベントループ遅延(s)	0.02	0.01	0.02

[9] API にアクセスするための特定の入り口

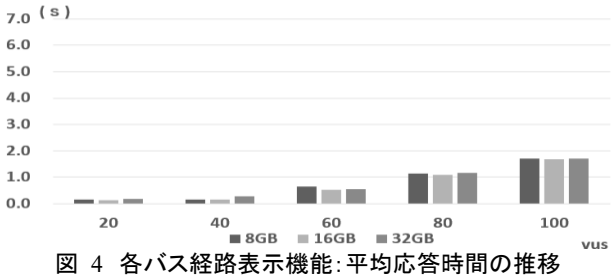

図 4 各バス経路表示機能:平均応答時間の推移

表 8「改善後システムでの地図スクロール操作」結果

評価項目	8GB	16GB	32GB
エラー発生率(%)	0.00	0.00	0.00
平均応答時間(s)	0.09	0.10	0.09
最大応答時間(s)	9.87	9.65	9.24
CPU使用率(%)	0.26	0.26	0.26
メモリ使用量(MB)	130.68	173.74	163.52
平均イベントループ遅延(s)	0.01	0.00	0.00

これらの結果から，機能ごとに負荷特性が大きく異なることが確認された．以下，各機能について考察する．

表 5に示すスポット推薦機能の結果では，エラー発生率が約0.1%，平均応答時間が0.07秒であった．同様に，表 8に示す地図スクロール操作の結果では，エラー発生率が0%，平均応答時間が約0.1秒であった．いずれの機能もサーバスペックによる差異は認められず，アクセス集中時においても実用上問題のない軽量な処理であるといえる．

一方，表 6 に示すすべてのバス表示機能の結果では，エラー発生率が約 6〜7%，平均応答時間が約 6 秒，メモリ使用量が約 350MB であり，他の機能と比較して高い負荷が確認された．サーバスペック間では，エラー発生率は 8GB で 7.50%，16GB で 6.43%，32GB で 6.25%となっており，高スペックであるほどエラー発生率が低下する傾向が認められた．また，図 2，図 3 から，同時アクセス数の増大に伴い，エラー発生率と応答時間が増大する傾向も確認された．API リクエストの HTTP ステータスコードやレスポンス内容をログ解析した結果，エラーの主因は時刻表データ取得の失敗であることが判明した．本システムの時刻表データは，約 1950 箇所のバス停に対する約 17000 件の発着時刻レコード（約 9MB）を含み，生成・転送に時間を要する．表 6 に示す最大応答時間が全サーバスペックで 60 秒となっていることから，同時アクセス数が増大すると時刻表データ取得処理がタイムアウト設定に達し，エラーとして計上されていたことが示唆される．

現行の会津バスの利用状況を考慮すると，最大でも瞬間的なピークは40人程度と見込まれている．図 2に示すように，同時アクセス数が40以下ではエラー発生率は低く，現行の利用規模においては改善後システムが安定して動作することが期待できる．一方で，同時アクセス数が増大した高負荷状態では，サーバ側で複数のバスに関するデータ（位置情報や時刻表等）の取得・統合が同時に発生し，応答データの作成と送信に要する時間が増大する．その結果，タイムアウトに達するリクエストが増え，エラーが発生したと考えられる．

表 7に示す各バスの経路表示機能の結果では，エラー発生率は0%であり，安定した動作が確認された．しかし，図 4から，同時アクセス数の増大に伴い応答時間が増大する傾向が確認され，一定の処理負荷が存在することが示唆された．

現行実装では，経路表示リクエストの都度，バス停データを取得する設計となっており，同時アクセス数が増大した場合には，バス停データ取得処理が重複して発生する．この結果，データ取得および転送に要する待機時間が増大し，表 7に示すとおり平均応答時間が1.7秒となったと考えられる．

以上により，改善後のシステムでは，機能ごとの負荷特性の違いが明確となった．表 5および表 8に示すスポット推薦機能と地図スクロール操作は軽量な処理である一方，表 6および図 2，図 3に示すすべてのバス表示機能は，高負荷となる傾向が顕著であり，今後さらなる最適化が必要であるといえる．

6. むすび

本研究では，Node.jsを用いたWebアプリを対象として，アクセス集中時における高負荷耐性の課題を明らかにし，その改善手法を検証した．検証対象として，実運用を想定したバスロケーションシステムを用い，負荷試験ツールk6およびClinic.js Doctorを使用して評価を行った．

既存システムでは，サーバのCPU使用率やメモリ使用量に余裕があるにもかかわらず，高負荷状態で著しい性能低下が確認された．また，サーバスペックを変更しても性能に大きな差が見られなかったことから，高負荷状態のボトルネックはハードウェア性能ではなく，システムの実装構造に起因するものであることが示唆された．

そこで，既存システムの構成を見直し，サーバ側処理とクライアント側処理を分離するとともに，初期表示時に行っていたHTMLの動的生成を静的配信へ変更した．その結果，初期表示におけるタイムアウトとエラーが解消され，アクセス集中時の高負荷耐性が向上した．

一方で，本研究では特定の改善手法および検証環境に限定した評価を行っており，高負荷要因を網羅的に解決したわけではない．今後の課題として，負荷の高い機能に対する個別最適化，データ取得と生成処理の効率化等が挙げられる．

謝辞

本研究は福島県会津地方振興局「生活交通」事業の「路線バスの利用しにくさの改善」の一環として取り組みました．また，ご多忙のところご協力いただきました，会津若松市様，会津乗合自動車株式会社様，AIOIΔ様に厚く御礼申し上げます．

参考文献

[1] 掌田津耶乃，Node.js 超入門第 4 版,秀和システム，2023．
[2] Node.js，Node.js の紹介，https://nodejs.org/ja/learn/getting-started/introduction-to-nodejs,(参照:2025-06-21)．
[3] Node.js，ブロッキングとノンブロッキングの概要，https://nodejs.org/ja/learn/asynchronous-work/overview-of-blocking-vs-non-blocking,(参照:2026-01-29)．
[4] Grafana Labs，”Metrics”，Grafana k6 Documentation，https://grafana.com/docs/k6/latest/using-k6/metrics/,(参照:2026-01-21)．
[5] 桐生実和，”バスの待ち時間活用のための利用者ニーズに基づくスポット推薦システムの開発 ―路線バスの利用改善策の提案―”，会津大学短期大学部 2023 年度経営情報コース卒業論文要旨集，2023．
[6] 鈴木皓太，”地方路線バスの利用促進のためのシステム開発 ―GTFS データを用いた総合的な情報提供システム―”，会津大学短期大学部 2023 年度経営情報コース卒業論文要旨集，2023．
[7] 黒須友香，“Webアプリのユーザビリティの改善案―地図表示を伴うインタラクティブコンテンツを対象として―”，会津大学短期大学部 2024 年度経営情報コース卒業論文要旨集，2024．
[8] 安田萌，“Web APIを実装したWebアプリの応答性向上―Node. jsを使用したバスロケーションシステムを用いて―”,会津大学短期大学部 2024 年度経営情報コース卒業論文要旨集，2024．