研究指導 中澤 真 教授

Web API を実装した Web アプリケーションの応答性向上 —Node.js を使用したバスロケーションシステムを用いて—

安田 萌

1. はじめに

近年，Webアプリケーション（以下Webアプリ）は多くの分野で活用され，情報収集やコミュニケーションの手段として必要不可欠な存在となっている．特に，Web APIを活用したシステムは，外部サービスとの連携を可能にし，より高度な機能を提供できる．その一方で，APIの呼び出しに伴うサーバとの通信処理が増加し，Webアプリの応答速度の低下が課題として挙げられる．

一般的に，Webアプリの応答性が低下すると，ユーザは操作の遅延を感じやすくなり，快適に利用することが難しくなる．応答性が低下する要因として，Web APIのレスポンス速度の遅延や，サーバとクライアント間の通信負荷が挙げられる．加えて，Node.jsによるWebサーバシステムを用いた場合，シングルスレッドによる処理の制約が影響し，リクエストの処理が遅延することがある．これらの要因が重なると，Webアプリの応答性が低下し，結果的にユーザビリティにも悪影響を及ぼす．

そこで，本研究ではGoogle Maps APIとNode.jsを用いたバスロケーションシステムを題材とし，システムの応答性向上のための具体的な改善手法を明らかにする．

2. Web アプリケーションの特徴と作成時の問題点

2.1 Web アプリケーションの特徴と問題点

現在，多くの人々に利用されるWebアプリは，従来の静的なWebサイトとは異なり，サーバとのやり取りによって動的にコンテンツを生成するため，ユーザはフォーム入力やボタン操作などを通じて情報を送信・更新できる[1]．また，ネイティブアプリケーション[1]とは異なり，特定の端末やOSに依存せず，インターネット接続環境さえあれば利用可能であるため，デバイスへのインストールが不要で，多様な環境でも同一の機能を提供しやすいという利点がある[2]．

一方で，Webアプリは利用時に常にサーバ・クライアント間での通信を必要とするため，表示されるまでに応答時間が長くなってしまう場合がある[2]．応答時間の悪化はユーザビリティを著しく低下させることになる．ユーザが帯域の狭い通信環境を利用している場合にも，一定水準のサービスを提供できるように，開発者が応答速度の向上に取り組むことが重要となる．

静的コンテンツであるWebサイトでは，画像のデータサイズやテキスト量の問題が応答性に影響を与えていた．これに対し，動的コンテンツであるWebアプリは，ユーザの操作に応じてサーバ・クライアント間の通信が随時発生し，それぞれの環境で様々な処理が行われる．そのため，画像やテキストのデータサイズだけでなく，通信環境や処理負荷も応答速度に大きく影響すると考えられる．

2.2 Web アプリケーションにおける応答性向上のための方法

Webアプリの応答性を低下させる要因として，データベース処理を含むアプリケーション内の各種処理が肥大化・複雑化していることが挙げられる．この応答性の課題に対して，これまでも改善アプローチが試みられてきた．涌井ら[3]は主にフロントエンドにおける文章要素の軽量化に焦点を当てて，応答性の向上を図っている．これによりテキストベースのWebアプリにおいて，大幅な処理時間の短縮が可能になることを示している．

2.3 Web API を実装した Web アプリケーションの応答性向上の課題

涌井らの研究では，HTMLやCSSからなるテキストをメインとしたWebアプリを用いていたが，近年のWebアプリでは，視認性を向上させるために，JavaScriptを用いて動的な表示や操作を取り入れたものが主流になってきている[4]．画像や地図を用いた表現やJavaScriptによるアニメーション表現が増え，単純なテキスト表現から，インタラクティブな要素を含む高度な表現が可能なシステムへと変化した．

それに加えて，Google Maps API[2]などの外部サービスであるWeb APIを積極的に活用するケースも増えている．これらのAPIを導入することで，Webアプリの機能や質を高めつつ，開発効率を向上させることが可能となる．しかし，外部サービスとの通信に伴う遅延が発生し，応答時間に影響を及ぼしてしまう場合もある．

Webアプリはリッチなコンテンツを提供すると同時に，ユーザ操作に対するインタラクティブ性実現のためにクライアントとサーバへの負担を増やしてしまうという問題も抱えている．クライアントは表示・実行に対しての処理だけでなく，複雑なデータ処理も要求されるようになり，システム全体の性能に大きな影響を与えるようになった．

このような背景から，従来のテキストベースのWebアプリにおける改善策とは異なり，Web APIの実装やサーバサイドとクライアントサイドの効率化が現代のWeb開発における重要な研究課題となっている．特に，多様な媒体と通信環境に対応しつつ，高速で安定したシステムを提供することが求められている．

---

[1] デバイスにインストールして使用するアプリケーション

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

そこで本研究では，上述のWebアプリの変化に合わせた，応答性向上のための改善策の提案を目的とする．

3. 研究に使用するWeb APIを実装したシステムの現状の評価

本研究では，桐生[5]・鈴木[6]が開発した会津若松市のバスロケーションシステムを題材に検証する．

桐生の研究では，ユーザの現在地をGPSで取得し，目的となるバス停から利用するバスの時刻表を選択することで，システムが待ち時間や移動時間を計算する．さらに，ユーザの状況を「地元/観光客」「空腹度」「疲労度」の3つの質問で把握し，推薦スポットを3段階に分けてマップ上に表示するシステムを開発した．このシステムは，バスの便数が少ない地域における，待ち時間の長さによる不便さを解決するものとなった．

鈴木の研究では，バス停をマップ上に配置し，選択されたバス停を発着するバスの位置情報をリアルタイムで把握できるようにした．従来のバスロケーションシステムでは，テキスト形式の遅延情報のみを提供していたが，このシステムでは地図上にバスの動きを視覚的に示すため，到着予定時刻を直感的に確認できるようになった．さらに，東西南北の4種類のアイコンを使い分けることでバスの進行方向を可視化し，バス選択時には通行経路上のバス停のみを線で結んで表示する機能を実装している．これらの機能により，バスを普段利用しないユーザでも適切なバスを容易に選択できるようになり，ユーザビリティの向上を図っている．

上記のシステムでは，地図によるバスロケーションシステムの表示を行うためにGoogle Maps APIを実装している．先に述べたように，Web APIは外部サーバとのやり取りの発生や地図情報の大量のデータ転送により，応答速度低下を招くことがある．また，JavaScriptをサーバサイドで実行するためのNode.jsは，OSのスレッドを使用する一般的な同時実行モデルとは対照的に，シングルスレッドで動作するため並列処理ができず，処理時間がかかる傾向にある[7]．

これらの影響を含む複数の要因が重なることで，上記のシステムでは初期表示に時間がかかり，ユーザが操作した際に，体感的にも遅いと感じる場合がある．本研究では，定量的に評価するため，Googleで提供されているLighthouse機能[3]を用いて測定を行った．Lighthouseではモバイルとデスクトップの評価が可能だが，本研究ではバスロケーションシステムの特性を考慮し，屋外での利用を想定してモバイル環境での計測を行った．計測の評価項目を以下に示す．

- First Contentful Paint（FCP）：ページの最初のコンテンツが表示されるまでの時間
- Largest Contentful Paint（LCP）：最大のコンテンツ要素[4]が表示されるまでの時間
- Total Blocking Time（TBT）：ページの操作可能になるまでの待ち時間[5]
- Speed Index（SI）：コンテンツが視覚的に表示される速度

桐生・鈴木のシステムの計測結果は，FCPが55.40秒，LCPが65.90秒，TBTが18.85秒，SIが55.40秒と，4項目すべてが低い評価となり，応答性に問題があることが確認できた．この状態では応答速度の改善が不可欠であり，涌井らの研究で行われた軽量化だけではなく，Web APIやNode.jsに合わせた改善方法が必要になる．そこで，本研究では既存システムを題材として応答性向上に有効な方法を探り，より一般的なWeb APIやNode.jsベースのWebアプリに適用可能な改善手法を明らかにする．

4. 応答性を向上させるための改善策

本章では，Web APIを利用するバスロケーションシステムの応答性を向上させるために検討・実装した4つの改善策と，今回の検証では効果が見られなかった2つの改善策について述べる．具体的に実装した改善策は以下の4つである．

1. 軽量ライブラリへの変換
2. Scriptの非同期処理
3. バス停表示をユーザ画面のみにする
4. サーバからのレスポンスをgzip圧縮して返す

なお，評価指標は前述の方法と同様，Lighthouseを主として計測し，応答性の効果を確認している．

4.1 軽量ライブラリへの変換

jQuery[6]はJavaScriptのライブラリ[8]であり，既存システムでも採用されていた．jQueryはDOM操作やAjax通信[7]などの機能を簡素化する一方で，ファイルサイズが86KBある．この容量は，特にモバイル環境での初期ロード時間に影響を与える可能性がある．そこで，jQueryの機能を維持してコードを変更せずに，より軽量なZepto.js[8]への移行を検討した．Zepto.jsは，モバイルブラウザに最適化されており，必要最小限の機能に絞っているため，ファイルサイズがjQueryの約3分の1の26KBである．実装にあたっては，Zepto.jsのjQueryとの互換性が完全ではないという特性を考慮し，Zepto.jsでサポートされていない部分を精査し，各端末における動作確認を実施したうえでシステムに実装した．

ライブラリ変更後の計測結果は，FCPが55.70秒，LCPが65.80秒，TBTが18.37秒，SIが55.70秒となり，数値上の大きな改善は見られなかった．しかし，この結果は単体での評価であり，後述するほかの改善策と組み合わせることで，システム全体のパフォーマンスに寄与する可能性がある．特に，モバイル環境でのメモリ使用量の削減や，ファイルサイズの縮小による二次的な効果が

---

[3] https://developer.chrome.com/docs/lighthouse?hl=ja

[4] 画面に映る範囲内で最も大きな画像やテキストブロック

[5] ユーザの操作に対して反応が得られるまでの待ち時間

[6] https://jquery.com/

[7] Webブラウザ内で非同期通信を行いながらインターフェイスの構築を行う技術

[8] https://zeptojs.com/

期待できるため，Zepto.jsへの移行を採用する．

4.2 Script の非同期処理

既存システムでは，各Scriptを同期的に呼び出していたため，Node.jsのシングルスレッドという特性上，一つのScriptのダウンロードと実行が完了するまでの間，ブラウザは描画処理を中断せざるを得なかった．その結果，ユーザが操作してもすぐに反応を得られず，応答が返ってくるまで待機を余儀なくされることになっていた．さらに，複数のライブラリや大きなコードを読み込む必要がある場合には，ページの初期表示が遅れる問題が生じていた．

そこで本研究では，Scriptを非同期的に処理することで，応答性の向上を図った．そのため，ライブラリとGoogle Maps APIの呼び出しをasyncで行うように変更し，各Scriptが他のScriptに依存せず，並行して動作できるようにした．

表 1 Scriptの非同期処理による応答性の変化

評価項目	変更前	変更後
FCP	55.40秒	54.10秒(-1.30)
LCP	65.90秒	55.10秒(-10.80)
TBT	18.85秒	3.02秒(-15.83)
SI	55.40秒	54.10秒(-1.30)

評価結果から，特にLCPとTBTで顕著な改善が見られた．また，軽量のライブラリを非同期的に読み込むことで，メインスレッドのブロッキングを最小限に抑えられると考え，改善案1（軽量ライブラリの変換）と2（Scriptの非同期処理）を併用した場合の効果も測定した（表 2）．

表 2 非同期処理と軽量ライブラリの併用による応答性の変化

評価項目	変更前	変更後
FCP	54.10秒	53.90秒(-0.20)
LCP	55.10秒	54.10秒(-1.00)
TBT	3.02秒	1.08秒(-1.94)
SI	54.10秒	53.90秒(-0.20)

評価の結果から，LCPとTBTにおいて1秒以上の短縮効果が確認され，併用した場合に応答性がより向上することが示された．これは，Zepto.jsのファイルサイズ削減のみでは限定的であったが，ボトルネックとなっていた同期処理によるブロッキングを解消したことで，評価指標に顕著な変化が見られたと考えられる．したがって，Scriptの非同期処理とZepto.jsの併用が，応答性向上に有効であることが確認できた．

4.3 バス停情報を地図表示範囲に限定

既存システムでは，初期表示時にすべてのバス停情報を一括でクライアントに送信していたため，不要なデータ転送が発生していた．そこで，初期表示時間にかかる時間をconsole.time()メソッドで計測したところ，約10秒を要していることが明らかになった．この問題を踏まえ，本研究ではバス停情報について地図表示範囲に限定して送信し，データ量を抑えることで，初期表示を高速化する手法を考えた．

まずクライアントサイドで地図の表示範囲情報を取得し，そのデータをサーバサイドに送信するように変更した．サーバ上では，受け取った範囲に含まれるバス停情報だけを抽出し，これをクライアントに返信するようにした．バス停情報の制限機能を追加した結果，Zepto.jsの非同期状態と組み合わせることで，従来は正常に動作していた地図表示のタイミングがずれてしまい，マップが表示されなくなる問題が発生した．そこで，ライブラリの呼び出しタイミングのみ同期処理に戻し，処理手順を制御することで，マップ表示の不具合を解消した．その結果，console.time()メソッドによる測定において初期のバス停表示に要する時間が約0.31秒となり，改善前と比較して約10秒短縮することができた．

表 3 バス停情報を地図表示範囲に限定した場合の応答性の変化

評価項目	変更前	変更後
FCP	53.90秒	54.80秒(+0.90)
LCP	54.10秒	57.70秒(+3.60)
TBT	1.08秒	0.66秒(-0.42)
SI	53.90秒	54.80秒(+0.90)

評価の結果から，ライブラリ処理のみ同期処理に戻したことによって，改善した数値が戻ってしまった項目もあるが，TBTでは約400ミリ秒の変化が見られた．ミリ秒単位ではあるが，TBTで1.08秒から0.66秒へと変更されたため，約半分の時間へ短縮されたことになる．

以上の結果から，バス停情報を地図表示範囲に限定して送信する変更は，初期表示とTBTの時間を減らす効果があることがわかり，応答性を向上させる工夫としては有意義であると考えられる．

4.4 サーバからのレスポンスを gzip 圧縮して返す

既存システムでは，サーバからクライアントへのレスポンスにおいて，データを無圧縮のまま送信していた．サイズが大きい状態で通信するため，約12MBあるバス停情報のJSONファイルもそのまま転送されるようになっていた．そこで本研究では，サーバからのレスポンスを圧縮し，サイズを小さくすることで応答性の向上を図った．

サイズを小さくして返すために，compressionミドルウェア[9]を使い，HTML，CSS，JavaScript，JSONなどのテキストファイルがgzip圧縮されるように変更した．

表 4 サーバレスポンスを gzip 圧縮した場合の応答性の変化

評価項目	変更前	変更後
FCP	54.80秒	4.80秒(-50.00)
LCP	57.70秒	7.70秒(-50.00)
TBT	0.66秒	0.66秒(-0.00)
SI	54.80秒	5.00秒(-49.80)

評価の結果から，FCP，LCP，SIのそれぞれで約50秒短縮された．ファイルを圧縮したことによって，通信量の減少に伴い読み込み時間が短くなったと考えられる．

以上の結果から，本研究のように比較的大きいサイズを扱うシステムでは，gzip圧縮を導入することが応答性の向上に影響を与えると考えられる．

[9] https://www.npmjs.com/package/compression

4.5 改善されなかった案

試行したが，応答性の改善が見られなかった案として，以下の2つが挙げられる．

1. 地図情報の制限
2. ソースコード内の空白・改行・コメントアウトの削除

地図情報の制限では，地図の表示範囲を県内に限定し，Google Maps標準の商業施設などの店舗情報を表示しないよう変更した．しかし，地図の描画処理はGoogle Maps APIを使用しているため，その処理は本システムの外部システムであるGoogle側で動的に処理される．それゆえ，この制限は応答性にほとんど寄与しないことが明らかになった．

次に，JavaScriptやCSSファイルのソースコード内から空白や改行，コメントアウトなどの不要な文字列を取り除くことで容量を削減し，応答性の向上を試みた[9]．具体的には，可読性のために設けられたインデントの削除やコードについての説明書きなどのコメントアウト部分を削除することである．

この手法では応答性で大きな変化が見られなかった．その理由としては，既存システムのJSファイルは87KBに過ぎず，削除できたデータサイズが約4KBの縮小にとどまったことが考えられる．

5. 最終評価

本研究では，Web APIを実装したWebアプリの応答性向上を目的として，軽量ライブラリへの変換，Scriptの非同期処理，バス停の表示個数制限，サーバレスポンスのgzip圧縮の４つの改善策を実装し，その結果をLighthouseで計測した．

表 5　Lighthouse による最終評価比較

評価項目	変更前	変更後
FCP	55.40秒	4.80秒(-50.60)
LCP	65.90秒	7.70秒(-58.20)
TBT	18.85秒	0.66秒(-18.19)
SI	55.40秒	5.00秒(-50.40)

評価の結果から，初期状態と比較して，FCP，LCP，SIの3項目では50秒以上の大幅な短縮を実現した．また，TBTについても，18.85秒から0.66秒へと改善され，ユーザが待たされる時間が顕著に減少した．

この改善は，特に初期表示の高速化に大きく寄与した．既存システムではバスロケーションシステムの表示までに時間がかかるため，フリーズしたのではないかと誤解を生む問題があったが，改善後のシステムでは即座に情報が表示されるようになり，ユーザがストレスを感じることなく利用できるようになった．

一方で，地図情報の制限や空白・改行の削除などの改善策は，ファイルサイズの削減にはつながったものの，応答性の向上には大きな効果が見られなかった．これは，本研究で扱ったシステムにおいて，ボトルネックがサーバとの通信やJavaScriptの同期処理にあったためであり，使用している情報やファイルサイズの大きさによっては，異なる効果が見られる可能性もある．

本研究の結果から，Web APIを利用するWebアプリにおいて，ライブラリの軽量化による初期ロード時間の短縮，gzip圧縮による通信量の削減，バス停情報の動的取得による不要なデータロードの抑制，Scriptの非同期処理による描画ブロッキングの回避が有効であることが確認できた．これらの手法は，本研究で用いたバスロケーションシステムに限らず，外部APIとの通信が多いWebアプリ全般に適用可能であり，応答性を重視するシステム開発全般で有効な方法であるといえる．

6. むすび

本研究では，Web APIやNode.jsを用いたWebアプリにおける応答性向上を目的とし，複数の改善策を提案した．既存のバスロケーションシステムでは，初期表示に時間がかかり，ユーザが操作した際に遅延を感じることでユーザビリティが低下するという課題があった．特にGoogle Maps APIを用いた地図表示やサーバとの通信処理がボトルネックとなり，実際の利用シーンにおいてはストレスを感じる要因となっていた．

この問題に対処するために，ライブラリの軽量化による初期ロード時間の短縮，gzip圧縮による通信量の削減，バス停情報の動的取得による不要なデータロードの抑制，Scriptの非同期処理による描画ブロッキングの回避を実施した．その結果，応答速度の大幅な短縮が確認され，応答性の向上につながることが示された．

一方で，今回導入した手法は，特定の環境下での評価に基づいており，異なるネットワーク条件やデバイスでの挙動については十分に検証されていない．今後は大規模なアクセス時を想定した負荷試験を実施するなど，実運用に耐えるシステム開発を目指す．

謝辞

本研究は，福島県会津地方振興局「会津DX日新館」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に厚く御礼申し上げます．

参考文献

[1] GMO おみせアプリ，Web サイトと Web アプリの違いについて解説，https://gmo-app.jp/column/6147.html，（参照 2024-06-19）．

[2] iRidge，ネイティブアプリとは？，https://iridge.jp/blog/202302/33552/，（参照 2024-06-19）．

[3] 涌井智寛　ほか，”Webアプリケーションのユーザ快適性向上を目指した一つのモデルの提案と解決”，情報処理学会研究報告 2010 年度 8 号，pp.1-8，2010.

[4] 張 恭瑞　ほか，” Web アプリケーションの UI 開発支援のためのテンプレートの抽出手法“，コンピュータ ソフトウェア 2016 年 33 巻 1 号，p.1_150-1_166，2016.

[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”，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”，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] Node.js，Node.js®とは，https://nodejs.org/ja/about，（参照 2025-02-01）．

[8] Jeremy L.Wagner，Web サイトパフォーマンス実践入門，翔泳社，2018.

[9] 窪田優,Web サーバ高速化教本，秀和システム，2019.