初めてのシングルページアプリケーション
Vue.js と Firebase で作るミニ Web サービス

渡邊達明 著

2 版 ザ・シメサバズ発行

目次

1. まえがき

本書を手にとっていただきありがとうございます！

まずは購入するかどうかを考えている方もいるかと思いますので、本書のターゲットとそうでない方を明記しておこうと思います。もし下記ターゲットに当てはまる場合は是非この本を手にしていただき「SPA がどういったものかつかめてきた！」「自身で作った Web サービスを使ってもらえる楽しさ」などを体験していただければ何よりです。

1.1. この本の目的・ターゲット

本書では以下のような方をターゲットとして執筆しています。主に簡単な Web サイトを作った事がある Web フロントエンド初心者の方が、本書を通じて SPA の基本を掴み、簡単な Web サービスが作れるようになることを目的としております。そしてそれらを抑えた上でさらに次の段階へステップアップするための足がかりになるような要素を各所に入れております。

- HTML,CSS,Javascript を利用して、簡単な Web サイトを作ったことがある人
- 複数のページや状態管理を Javascript 使って自力で行い、ごちゃついてしまい消耗している人
- 普段デザイナーとしてマークアップしており、フロントエンドの新しめな環境でのサイト構築を経験してみたい人
- シングルページアプリーケーション、Vue.js ってものを触ってみたい、またはそれらの挫折経験がある人
- やったことないけどとにかく Web サービスを作ってみたい人
- Firebase の Web 版を使ってみたい人、どんな事が出来るのか知りたい人
- 普段サーバーサイドやネイティブアプリを作っていて、Web フロントエンドをさらっと触ってみたい人

また、最終的にどんな Web サービスが作れるか、というサンプルを公開しております。今回のプロジェクト名は ”MyMarkdown” とし、本書で進行していく上でこの名称が出てきた場合は、都度ご自身のオリジナルのプロジェクト名に置き換えつつ進めてください。

サンプルサービスは本書の内容にプラスして見た目や使いやすさを向上させるために多少手を加えたものですが、元の機能面ではあまり差はありません。

MyMarkdown
https://mymarkdown.firebaseapp.com

機能的には Google アカウントでログインし、マークダウン形式でプライベートなメモが書ける Web サービスとなっております。

本サイトのソースコードについてもこちらに公開しておきますので、途中詰まってしまった時などにご利用ください。

https://github.com/nabettu/mymarkdown

1.2. 本書であまり触れない部分

本書にでは主に「Vue.js と firebase を使って簡単な Web サービスを作ってみる」という事を目的としているため、各技術の詳細や網羅的な内容は収まりきらないため省略しております。

- React.js や Angular.js などの他のフレームワーク・ライブラリとの差分について
- Vue.js の場合は Vuex などを用いて状態管理を行うのが SPA 開発では一般的ですが、今回は入門としてその部分は触れずに、コンポーネントをまたいだ状態管理は行いません
- Firebase の中でも Functions や 2018 年 1 月現在ベータ版である Cloud Firestore、アプリ SDK などの内容については触れません。本書では Authentication、Hosting や Realtime Database についてのみ利用します
- css についての詳細なテクニックや、PC/SP での表示切り替えなどはあまり触れません。本書の手順で Scss 自体は利用します。
- Nuxt.js などを用いたサーバーサイドレンダリングについてや PWA 化、パフォーマンス・チューニング等について
- webpack でのコンパイル等についての詳細や機能追加
- セキュリティやテストについての内容

ある程度省略はしますが「この本を終えた後、何をやればいいかわからない」ということにならないよう、できるだけ各項目について名称やざっくりした説明を付け加えておきます。

また、本書の最後には「強くてニューゲーム」として、追加で実装すると良い機能などを羅列しておきますので、その後のステップへ進む手がかりにしていただければと思います。

1.3. 本書の構成と対応環境について

目次を見ていただければ本書の大体の流れが想像出来ると思います。

“ 前置きはいいのでさっさと開発を始めたい ” という人は 3 章から読んでいただければと思います。

本書で開発を進める環境は MacOS を前提としておりますが、Windows でもセットアップ以

外はほぼ問題なく同じように進められるはずです。コマンドを利用する場合は Mac ではターミナル (Terminal.app) ですが Windows の方は (cmd.exe) コマンドプロンプトを利用していただき、都度コマンドは対応するものに読み替えていただければと思います。

コマンドを利用する際には下記の記述で行います。($ は打たなくても大丈夫です。)

```
$ ここにコマンドが入ります。
```

本書ではそれほど利用する場面は多くありませんが、黒い画面が苦手だった方はこれを機に利用してみていただければと思います。使いこなせてくると非常に便利です。

今回執筆時に利用したローカルでの Node.js のバージョンは 6.1 です。バージョン差異によってエラーが発生するような場合には nodebrew や nodist を利用して、できるだけバージョンをあわせていただければと思います。

Node.js についてはご自身の PC へローカルで動作するよう事前にインストールしておいていただければと思います。また、執筆の際に利用した npm のバージョンの詳細についてはサンプルコードのリポジトリの中の package.json を見ていただければと思います。

本書で**コラム的に用語の説明が個別で必要そうな場合には下記のような形で記載**していきますので、すでにご存知の方は読み飛ばしていただければと思います。

npm って？

npm とは Node.js の**”パッケージ管理ツール”**です。

ここで言うパッケージとは Node.js で利用出来る便利な機能が入ったライブラリやツールをまとめて管理するためのものです。Node.js をインストールしておくと npm コマンドですぐ利用出来るようになります。

その際に管理データを package.json というファイルにまとめておくことができ、その中に「このプロジェクトでは Vue.js のバージョン 2.5 を使います」などの情報が記載されています。

$ npm install

というコマンドを打つと、そのディレクトリにある package.json を参照して node_modules というディレクトリにそのライブラリ郡をダウンロードしてくるといった機能があります。

ファイルの編集が必要な部分については、編集する行についてのみ下記のように記載します。

▼ /index.html

```
<!DOCTYPE html>
<html lang="ja">
  <head>
```

ファイルの一部を書き換える場合はこのように変更部分を**太字**で記載しています。ファイル全体を記載する際にはそのまま載せています。

初心者でない、すでに他のフレームワーク等を利用した事がある方や、**もう一歩進んだ事をやりたい方などで本書を読み進めている方**については下記のような形で書き加えておきますので、是非挑戦していただければと思います。

ちょい足しポイント

このような形で、さらなる高みを目指す際のヒントをところどころに記載しておきますので、余裕があれば是非試してみてください。

1.4. 免責事項

本書を参考にして制作する内容については、ソフトウェアのバージョン違い等でお手元の環境ではスムーズに動作しないこともあるかもしれませんが、保証はいたしかねます。

不正確な部分や誤認、セキュリティホールになりうる内容などありましたら、お知らせ頂けますと幸いです。電子版については適宜改訂を行います。

ユーザー登録を行って利用するサービスを開発するため、もしご自身で制作した Web サービスを公開する場合は情報の取扱については十分にご注意ください。

2. サーバーレスシングルページアプリケーションの基本

まずシングルページアプリケーション（長いので、以下 SPA と記載します。温泉ではない）とはどういったものかという事から説明します。また、実際現場で求められているのは既存の Rails などで開発された Web サービスへ SPA を追加でフロントエンド部分のみを差し替える事だったりしますが、、、本書ではサーバーレスで SPA を使って Web サービスをゼロから構築してみる事としますので、既存サービスへの置き換えは別途調べていただければと思います。

2.1. SPA とフレームワーク（Vue.js の紹介）

SPA とはシングルページアプリケーションの事でありますが、何を持って " シングルページ " なアプリケーションなのでしょう。 少し歴史的な部分も含めて簡単に説明します。

2.1.1. 何がシングルページ？

以前は Web サイト上の情報はすべてページを切り替える事でしか出来ていませんでした。ページの URL やセッションの情報に紐付き、サーバーから出力する HTML を変更することでコンテンツを切り替えていました。

その後ブラウザの技術が進み、Ajax という技術を使ってブラウザ上で Javascript を使ってサーバーと情報をやり取りする事で**ページ遷移をせずに情報を更新出来る**ことが可能になりました。

また、pushState という機能がブラウザに追加されました。それによりブラウザ API を使って Javascript から URL を動的に変更出来るようになったことで、今までとは逆で**ページ内容に紐付けて URL を管理出来る**ようになりました。

これらを組み合わせる事によって**サーバーから返す HTML は常に同じでも、URL やセッションの状態に応じてサイトの内容を切り替えることが可能**になりました。 これが**シングルページ**と呼ばれる理由です。

※とはいえ昨今は SPA を採用しているサイトでもパフォーマンス向上のためにサーバーで返す HTML を切り替えていることがほとんどです。これを**サーバーサイドレンダリング**といいます。

2.1.2. SPA フレームワーク使うとなにがいいの？

SPA を作る上では先ほどの、" ページの再読み込みが無くともサイトの内容を切り替えること " が重要です。その為には URL やセッションにページの内容を記録しておくような機能、それらに応じて適切なデータを返すといった仕組みが必要になってきます。その周辺の機能を使いやす

くまとめたものがSPAフレームワークとなっています。

しかし「おなじみのjQueryでそんな機能くらい作れるのでは？」と思った方もいると思います。実際jQueryだけを使って、頑張ってSPAフレームワークと同じような動作をするコードを書くことは可能です。可能ですが、基本的にjQueryでは状態をHTML要素に持たせる事になり、コードが煩雑になってしまい、大規模なサイトで整合性を保つようなサイトを作ることは非常に難しいです。

フレームワークを利用することで複数人で開発する際や、サイトが大規模になった場合に非常にメリットが大きく、初心者でもサイトの状態管理を行うことが容易になるなどのメリットがあります。

今回はそんなSPAフレームワークの中でも比較的初心者でも扱いやすいVue.jsにフォーカスし、学習していこうと思います。

2.1.3. コンポーネント指向

SPAでの開発を行う上で欠かせないのがコンポーネント指向と呼ばれる考え方です。

SPAではURLやキャッシュの状態に応じてコンテンツを切り替えるという話をしましたが、その**コンテンツの管理が煩雑になりやすい**ものの一つです。

それを整理するために”コンポーネント”という単位にWebサイトのパーツを切り分け、個別に動作するような物としておくことで再利用性を高めたり、重複を防いだりします。

2.2. サーバーレスってなに？

SPAについての説明の次はサーバーレスについてです。

まずサーバーレスと一口に言っても様々ですが、本書ではサーバーインフラのリソースを気にしなくてもサーバーサイドの処理が出来る環境を利用することで、今までの**サーバー構築で考慮していた一部分を考慮する必要がなくなる**という意味で利用します。「サーバーリソースを使わない」わけではないです。

そして今回利用するFirebaseは、BaaSと呼ばれる”バックエンドの処理を肩代わりしてくれるサービス”であり、これを利用することによって比較的簡単にサーバーレスでサービスを開発することが可能になります。たとえユーザーが一度に数万人訪れたとしても、Firebaseに課金すれば自身でサーバーのリソースを気にする必要はありません（お財布は気にする必要がありますが）。

他にもサーバーレスを実現するためのサービスは多々あり、例えばAWS lambda等を利用することで今回と同じような構成をサーバーレスで開発することはもちろん可能ですが、初心者には少しハードルが高いです。その代わり自分でカスタマイズ出来る範囲が多かったり同じアクセス・データ量でも安く済むなど、メリットデメリットはありますので、もし業務で構築を求められたら色々と比較してみるとよいでしょう。

2.3. Firebase は何が出来る？

そんな Firebase ですが、今まで Web サービスを作る上でサーバーサイドプログラムが必要だった部分の一部が、コードを書かずに利用できます。

具体的に今回利用する機能としては以下の３つです。

Hosting：静的サイトのホスティング及び SSL 対応

Authentication：Google アカウントを利用してログイン、アカウント情報の取得

Realtime Database：NoSQL なデータベースの読み書き

それぞれ細かい利用方法は進めながら説明いたしますが、これらの機能を備えているため、他に何も準備しなくとも Web サービスを公開まで開発することが可能になっています。

今回利用する範囲ではフロントエンドの知識だけで開発することが可能であり、Firebase を利用することでサービスを開発することに集中することができます。

2.4. 今回そのままの構成が本格的な Web サービスで使える？

少ない機能でシンプルな、比較的小さい Web サービス運用なら十分使えるレベルです。

Facebook のようなサービスを作るというなら厳しいですが、そういった大規模 Web サービスを見据えていたとしても、モックとして今回のような構成で使う分には Firebase には十分で豊富な機能があります。

ある程度大規模になるようであれば、Realtime DB はリレーショナルでないので、知識が少ないとデータ管理が煩雑になりがちです。今現在 β 版ですがそういった欠点を補う Cloud Firestore という代替 DB が開発中です。

また、Firebase はスケーラブルではありますが、大量のユーザーやトラフィックを扱うとなると、運用にかかる費用はそのためのバックエンド開発を自前で行ったほうが運用コストは安い傾向があります。

3. 開発環境のセットアップとデプロイまでの流れ

おまたせいたしました！前置きが長くなってしまいましたが、ここから晴れて開発を始めます。ゼロからすべてのファイルを作成するのは大変なので、まずは今回開発する Web サービスの骨組みとなるテンプレートをダウンロードしてきます。

Vue.js ではプロジェクトを始める際の公式テンプレートが用意されており、そちらを利用することで比較的簡単に環境構築を行うことが出来ます。

今回は webpack を利用し、シンプルな構成でサイト構築を目的とするテンプレートを利用します。

> **ちょい足しポイント**
>
> 下記の webpack-simple テンプレートではない普通の webpack テンプレートを使って、unit テスト等を書いてテスト駆動で進めてみてください。

3.1. vue-temples のダウンロードとセットアップ

まずは下記のリポジトリのテンプレートを利用するために、PC に vue-cli というコマンドラインから vue のテンプレートをダウンロード出来るツールをインストールします。

https://github.com/vuejs-templates/webpack-simple/

コンソールを開いて下記のコマンドを実行して、vue-cli をインストールします。

```
$ npm install -g vue-cli
```

無事にインストールが出来たら下記コマンドでヘルプが表示出来ますので試してみてください。

```
$ vue -h
```

その後、作業するフォルダに移動して、テンプレートをダウンロードするための、下記コマンドを実行してください。実行すると、いくつか質問されますが、最後の User sass? 以外はそのまま enter で大丈夫です。sass のみ y を入力してください。

本書内でのプロジェクト名は "MyMarkdown" として進めますので、ご自身のプロジェクト名に適宜置き換えてください。

```
$ vue init webpack-simple mymarkdown
? Project name mymarkdown
? Project description A Vue.js project
? Author username <name@example.com>
? License MIT
? Use sass? Yes
```

※リポジトリの TOP からダウンロードして利用してもいいですが、package.json 等で {{#sass}} 〜 {{/sass}} と表記されている部分や {{name}} となっている部分を修正して使ってください。

今回使うテンプレートではあまり変わりませんが、他のテンプレートを利用する場合には vue-cli を利用することで初期設定時にどのモジュールを入れるのか等を細かく指定できますので余裕があれば触ってみてください。

手動ダウンロードの場合でも vue-cli の場合でも、無事にローカルにテンプレートをダウンロード出来ましたら、

```
$ cd mymarkdown
$ npm install
$ npm run dev
```

をそれぞれ実行してください。

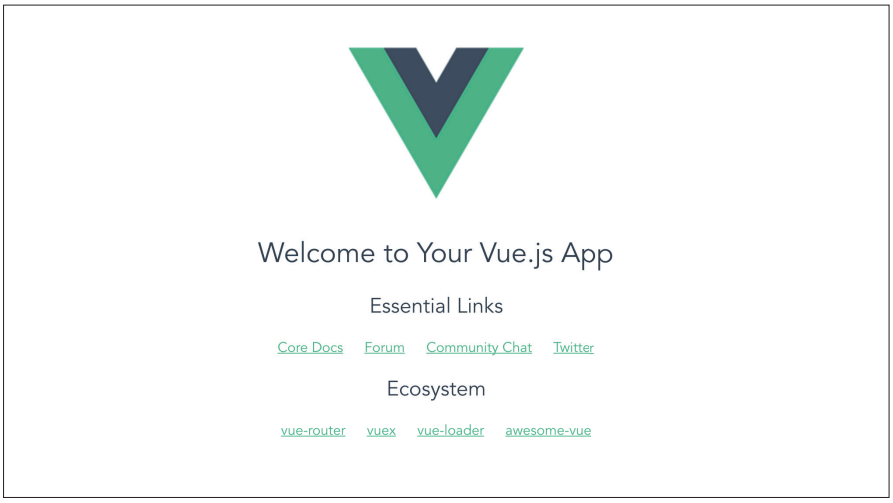


図 vue-template 実行画面

このような画面がブラウザ上で表示されましたでしょうか？

おめでとうございます。これであなたも Vue.js を使って Web サイトを表示することが出来ました。

もしエラーが表示されてしまい、セットアップがうまく進まなかった場合はエラー文言でそのまま Google 検索を行うと、同じ問題にあたっている方の情報が見つかる場合が多いと思いますので、参考にして解決していただければと思います。

npm-scripts とは

npm では npm-scripts と呼ばれる機能があり、先程

```
$ npm run dev
```

というコマンドで利用しています。

このコマンド自体は package.json の中の scripts という欄に実行出来るタスクを登録しておいて、npm から呼び出しが出来ます。

今回のタスクだと、

```
$ cross-env NODE_ENV=development webpack-dev-server --open --hot
```

という長いコマンドも短い名前で実行出来ます。

さらに自分でターミナルを実行する際には、グローバル環境に利用するコマンドをインストールする必要がありますが、npm-scripts ならそのプロジェクトの npm を利用するためインストールの必要がありません。（今回だと webpack-dev-server がそれにあたります。）

ただ、２つ以上のことをまとめて実行したい、並列処理直列処理を使い分けたい等やりたい事が増えて来た場合は gulp などのタスクランナーを利用すると、package.json と別ファイルで各タスクを定義出来るため、プロジェクトでやりたい事によって使い分けると良いでしょう。

では試しに内容の変更を行ってみましょう。

▼ /src/App.vue

```
return {
  msg: 'Welcome to MyMarkdown'
}
```

TOP ページの文言が変更されましたでしょうか？この時ブラウザを開いたままにしておくとよく分かるのですが、**ファイルを変更した部分のみが自動的に変更され、リロードがされない**と思います。こちらは Webpack の提供する **Hot Module Replacement** という機能でして、画面全体のリロードなしに変更してくれる便利な機能です。

https://webpack.js.org/concepts/hot-module-replacement/

また、App.vue ファイルという見慣れない拡張子のファイルを編集しましたが、これは Vue.js で利用するコンポーネント（ページそのものやサイトで利用するパーツの事）単位毎に ”HTML・Javascript・CSS” をファイル一つにまとめて書ける形式になります。

この形式は**単一ファイルコンポーネント**といって詳細は公式の以下のページにあるので、一読しておくと理解がより深まります。英語では SFC（Single File Components）と呼ばれますが、慶應義塾大学湘南藤沢キャンパスではありません。

https://jp.vuejs.org/v2/guide/single-file-components.html

ファイル一つにそのコンポーネントの情報が網羅され、サイト編集時にそこだけで簡潔するため見通しは良いですが、一つのコンポーネントに情報を詰め込みすぎると結局取り回しが悪くなってしまうので注意も必要です。

3.2. SFC でのコンポーネントの内容について

SFC の中身は template と script と style の３つの要素で出来ています。それぞれの要素につい

て詳細を見ていきましょう。

template

そのコンポーネントが表示する HTML を記載します。また、その html 要素に紐付いてクリックされたら何を実行するかなどの情報も @click=”hoge” のような形で template 内の要素に付与して定義します。」 {{ msg }} のような形で、後述する script で定義された コンポーネントの data を文字列として表示することも可能です。

基本的に Vue.js が行うことは data を html に整形して出力するのが主な仕事だと思ってもらってかまいません。

script

文字通り script としてそのコンポーネント内で処理するプログラムや template で表示するためのデータを記載します。

data に関しては文字通りそのコンポーネント内で利用するデータを格納する関数です。ここでは必ずオブジェクトを返却するようにしてください。その返却するオブジェクトに自由に変数を追加することで、HTML に内容を表示することが出来ます。

細かい各種の機能は利用する際に説明しますが、methods にそのコンポーネントで実行するメソッドを定義しておいて template で呼び出す部分を定義したり、他のコンポーネントを読み込んでおいてそれを template で呼び出したりなどもここに記述します。

style

各コンポーネントで利用する CSS を記述します。もしモジュール化してすべての CSS を全コンポーネントで共通化するのであれば必要ないですが、このコンポーネントでのみ適用される CSS を記述したい場合に scoped という機能があります。詳しくは 5.1.3 で説明します。

webpack って何？

webpack とはモジュールバンドラーと呼ばれ、js ファイルや CSS ファイル、設定によっては画像ファイルなども取り込み一つのファイルにまとめることが出来るツールです。

設定ファイルは webpack.config.js という名称で、今回利用するテンプレートでもルートに格納されています。そのファイル内でどのファイルをどのような設定で読み込むかが記載されています。

npm でライブラリ等をダウンロードしてきましたが、そのファイルはこの webpack を使い読み込みます。

例えば、/src/main.js のはじめの行では

import Vue from 'vue'

と書かれていますが、npm を使ってインストールしたライブラリ等は import 文を使い、この from 部分に記載することで利用することが出来ます。

import 文自体は実際にはまだブラウザ標準で動く機能ではなく、この場合は babel というトランスコンパイラ（トランスパイラ）を利用して**「古いブラウザが対応していない新しい書き方をして書いた javascript を、古いブラウザでも動作する形式に変換」**しています。

参考 URL
https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Statements/import

3.3. Firebase のセットアップとデプロイ

さて、無事にローカルで Vue.js を使って Web サイトを構築することが出来ました。次は Firebase を使って、いよいよ Web サイトをデプロイしてみます。(デプロイとはサイトを他の人が利用可能な状態に置くことです)

「え？もうデプロイするの？」そう思った方も少なくないかもしれません。安心してください、デプロイはしますがこの段階でローンチして不特定多数の人に見てもらおうというわけではありません。

”完成”するまでデプロイはしないものと思っている方もいるのではないでしょうか。Web サービスに”完成”などという段階は存在しません。Web サービスにとってローンチは、数あるマイルストーンの一つにすぎないのです。

今回すぐにデプロイする理由としては

デプロイしてみてローカルと本番で環境が違う事で、初めて見えてくる問題なども、ここで先に解決しておけます。

そもそもデプロイ出来なければローンチは出来ません。デプロイまでのステップを踏んでさえおけば準備が整えばすぐででもローンチが可能になります。

見える形にしておけば、友人などに途中経過を見せてフィードバック等をもらいやすいです。それによってモチベーションの低下を防げます。

もちろんデプロイはお楽しみにとっておいて頂いて、最後にドカンと行って頂いてもかまいません。

ということで、まずは Firebase のアカウントを作成するところからスタートしましょう。
https://firebase.google.com/

「使ってみる」ボタンの後、Firebase へようこそという画面で「プロジェクトの追加」を押してご自身のプロジェクトをつくり始めましょう。

図 プロジェクト ID の入力画面

この時プロジェクト名は管理画面上の名称ですが、プロジェクト ID は全ユーザーのプロジェクト内で固有の ID になります。そして今からデプロイするサイトの URL も、https:// プロジェクト ID.firebaseapp.com という形になります。（後から ID の変更は出来ないですが、公開前であれば別なプロジェクトとして作り直せます）

作成後「ウェブアプリに Firebase を追加」を押下してください。

図 .3 ウェブサイトに追加ボタン

モーダルが表示されているかと思いますがそこで表示されるコードをコピーし、mymarkdown ディレクトリの直下にある index.html 内にペーストしてください。その際**すでにある script タグの一行上にペースト**しましょう。

図 .4 HTML に貼り付けるコード

変候後の html は下記のようになります。

▼ /index.html

```
<!DOCTYPE html>
  <html lang="en">
  <head>
```

```
    <meta charset="utf-8">
    <title>mymarkdown</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="https://www.gstatic.com/firebasejs/4.8.2/firebase.js"></script>
    <script>
    // Initialize Firebase
    var config = {
      apiKey: "AIzaSyCiaxL8JBwL8deaJaOVLfAwF1yRWiiawJM",
      authDomain: "mymarkdown.firebaseapp.com",
      databaseURL: "https://mymarkdown.firebaseio.com",
      projectId: "mymarkdown",
      storageBucket: "mymarkdown.appspot.com",
      messagingSenderId: "619528626812"
    };
    firebase.initializeApp(config);
  </script>
  <script src="/dist/build.js"></script>
</body>
</html>
```

これらのタグは Web サイト上で Firebase のデータを扱ったり、SNS ログインするための命令を実行するためのものです。そしてこの情報はサイトに埋め込むため、ユーザーに公開されてしまうことになりますが、権限管理等は別の箇所で行うため、このコードは公開されていても問題ありません。詳細は 6.4 に記載します。

また Firebase の管理画面に戻り、今度は左のタブの Hosting というメニューをクリックしてください。この場合のホスティングとは index.html などをデータを保存し Web サイトの形で公開出来るようにしてもらえる機能のことになります。

「使ってみる」をクリックし、出てくるモーダルの命令をそのまま実行してみましょう。ここではお使いの PC のどのプロジェクト上でも使えるように firebase-tools をインストールします（グローバルインストール）。

※さきほどのプロジェクトが起動したままの場合は Control + c で “npm run dev” で動いていたプロジェクトを停止してから実行してください。今後もコンソールで動作しているプログラムを停止するコマンドなので覚えておいてください。

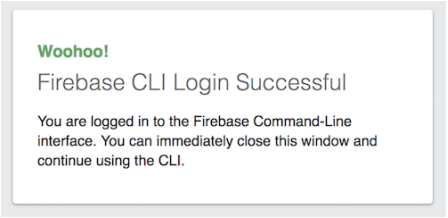
```
$ npm install -g firebase-tools
```

インストール出来たら、そのまま②の内容に進みます。

```
$ firebase login
```

実行するとブラウザが開き、この様な画面が表示されます。

OK を押して、



このような画面が表示されたら完了です。

こちらは自身の firebase プロジェクトの編集を、コンソールで操作している firebase-cli のアプリケーションに許可する手続きになります。

次はまたコンソールに戻り次のコマンドを実行します。

```
$ firebase init
```

init 時には初めにどの機能を利用するのか聞かれるため、カーソルを操作して **Hosting にカーソルを合わせて、スペースキーを押して選択状態にして Enter キー**、その後作成したプロジェクトを選択します。

「What do you want to use as your public directory?」
と、表示され次に公開に利用するディレクトリ名の入力が求められるため **dist と入力**してください。

「Configure as a single-page app (rewrite all urls to /index.html)?」
その後、公開するページは index.html だけかどうかを聞かれるので、そのままエンター (No) で進めてください。

すると自動的にプロジェクトのルートに
.firebaserc
firebase.json
の２ファイルが作成されていると思います。そこへ firebase で利用するプロジェクトの設定が記載されていきます。

次にホスティングしてもらうための準備として、テンプレートのままだと各種パスが開発用のままだったりするので、以下の２つのファイルを変更します。

▼ /webpack.config.js

```
8   publicPath: '/',
〜〜〜
78   devServer: {
79   +   contentBase: 'dist',
80   historyApiFallback: true,
〜〜〜
100   sourceMap: false,
```

※ Firebase の無料枠ではアップロード容量制限があるため、SourceMap が true の場合 map ファイルで容量オーバーの可能性があるためオフにします。

/dist/index.html が firebase-cli によって自動で作成されているため削除して、

▼ /index.html → /dist/index.html へ移動しつつ変更

```
21  </script>
22  <script src="./build.js"></script>
23  </body>
```

※ dist ディレクトリは標準で .gitignore に記載されています。本書では変更箇所を少なくするために dist ディレクトリに index.html を格納していますので git 管理する際にはご注意ください。

各種設定変更後に、デプロイするファイルを作成するために以下のコマンドを実行します。

```
$ npm run build
```

build が成功したら、ついにデプロイです。満を持して下記のコマンドを実行しましょう。

```
$ firebase deploy
```

成功した場合は “Deploy complete!” と表示され、公開 URL が表示されていると思います。アクセスするとローカルで作成していたものと同じ物が表示されているでしょうか。「build → deploy」でデプロイの手順は以上です。

今後本書を進める上で、区切りのいいタイミングがあったら是非デプロイしながら進めてみてください！

ちょい足しポイント

今回デプロイはローカルで行いましたが、もちろん CI に任せても OK です。その際 CI 用に firebase 用の token を発行することが出来るので、それを CI のコンテナにもたせ、firebase-tool をインストールし deploy コマンドを実行させてみてください。

4. Google アカウントでのユーザー登録と、ログイン状態の判別

Firebase の Hosting を利用できたので、次は Authentication を利用してユーザ登録・ログイン機能を利用してみましょう。その際に、ログインしている時としていない時で別な内容を表示するようにします。

4.1. components を作成し表示する

SPA ではコンポーネント単位でサイトの見た目を管理するという説明はすでに行っていますが、そのコンポーネントを作ることから初めてみましょう。

src ディレクトリの中に更に components ディレクトリを作成します。ファイルの場所は好みもありますので自由ですが、本書では今後作る .vue ファイルはすべてこのディレクトリに格納することとします。

ログインしていないときに表示する Home.vue と、ログインしている際にはメモ帳として Editor.vue を表示するのものとして新規作成し、以下の内容に編集します。

▼ /src/components/Home.vue

```
<template>
  <div id="home">
    <h1>{{ msg }}</h1>
    <button>Google アカウントでログイン </button>
  </div>
</template>

<script>
export default {
  name: 'home',
  data () {
    return {
      msg: 'Welcome to MyMarkdown'
    }
  }
}
</script>
```

▼ /src/components/Editor.vue

```
<template>
  <div class="editor">
    <h1> エディター画面 </h1>
  </div>
```

```
</template>

<script>
export default {
  name: 'editor',
  data () {
    return {
    }
  }
}
</script>

<style lang="scss">
</style>
```

双方中身は特にないコンポーネントとして定義だけしておきます。次に App.vue でそれぞれを読み込み、isLogin の状態に応じて表示を分けます。デフォルトで入っていた内容は削除します。

▼ /src/App.vue

```
<template>
<div id="app">
  <Home v-if="!isLogin"></Home>
  <Editor v-if="isLogin"></Editor>
</div>
</template>

<script>
import Home from './components/Home.vue';
import Editor from './components/Editor.vue';

export default {
  name: 'app',
  data() {
    return {
      isLogin: false
    }
  },
  components: {
    'Home': Home,
    'Editor': Editor,
  },
}
</script>
```

まずは別なコンポーネントの読み込みですが、import で始まる部分で相対パスで vue ファイルを読み込みます。その後、template 内で利用したいコンポーネントについては components 内で { tag 名 : 読み込んだ vue ファイル } という形式で定義しておきます。これで初めて template 内でコンポーネントの読み込みが行えます。

その後 template 内では HTML のタグのような形で定義したコンポーネントを読み出せます。もちろん template は HTML を記載する場所なので、既存の HTML のタグと同じ名称（例えば nav や section のような名称）

ではコンポーネントを作成出来ませんのでご注意ください。

次に v-if=”!isLogin” という記述について説明します。こちらは**条件付きレンダリング**と言って、v-if の場合はこのコンポーネントを表示するかどうかを中に入れた条件に応じて決めます。この場合は data で定義した isLogin というログイン状態を判別する変数に応じて表示します。試しに false を true に変更していただければエディター画面が表示されると思います。

Vue.js には v-else という条件付きレンダリングも存在していて、それを使うと一つ上の if の条件に合わなかった場合にそちらをレンダリングする指定もできます。注意点として、v-else は要素の順番が変わると意味が違ってしまい、条件だけでなく順序も考慮する必要も出てきてしまいます。そのためここでは使っていません。

あとは、この isLogin にログイン情報が入るようにすればログイン状態に応じて表示を切り替えることが出来るようになります。

ちょい足しポイント

今回は App.vue でまるごとコンポーネントの表示を切り替えていますが、通常のアプリケーションはもっとたくさんの表示すべきページがあり、なおかつ URL が分かれていてリロードしても状態が保持されていることが望ましいです。

そんな機能を実装する際には是非公式の Vue-router をお試しください。さくっと導入でき、ルーティングの管理をまるっと引き受けてくれるでしょう。

4.2. Firebase でログインの設定

次に Firebase の管理画面で Authentication タブを開きます。Firebase では Twitter や Facebook でのログインも対応していますが、それらは別でその SNS 側で開発者登録を行う必要があるため、今回は Google アカウントでのログインを実装します。

ログイン方法のタブを開き、Google をクリック、「有効にする」ボタンをクリックするだけで完了です。認証については下にスクロールすると、認証済みドメインを設定することも出来ます。例えば独自ドメインを取得した場合はこちらに追記が必要です。

4.3. Google ログインの実装

いよいよログインです。先程の Home.vue を開き。以下のように編集します。

▼ /src/components/Home.vue

```
<template>
  <div id="home">
    <h1>{{ msg }}</h1>
    <button @click="googleLogin">Google アカウントでログイン </button>
  </div>
</template>

<script>
export default {
  name: 'home',
  data () {
    return {
      msg: 'Welcome to MyMarkdown'
    }
  },
  methods: {
    googleLogin: function() {
      firebase.auth().signInWithRedirect(new firebase.auth.GoogleAuthProvider());
    }
  }
}
</script>
```

作成したボタンをクリックして、Google アカウントの認証に遷移したと思います。ここでアプリを承認すればログインしたことになります。

ここで初めて出てきた @click と methods について説明します。@click は HTML の onClick のような形で、その要素をクリックすると実行する命令を記載できます。onClick では javascript をそのまま記載出来ましたが、Vue.js では呼び出す関数を記載します。

そこで呼び出される googleLogin という関数は script 内の methods に記載されています。コンポーネント内で呼び出す関数はこのように methods 内に記載しておくことで利用できます。method 内で別 method を呼び出す際には this. メソッド名で呼び出しが出来ますが、function() の記載を上記の例の通りにしておく必要があります。ES6 で追加された Arrow Function の形 (`() => {}`) で記載すると this の内容が変わってしまうので注意が必要です。

ちょい足しポイント

今回は Google ログインを利用しましたが、サービスの内容によっては Twitter のほうが相性がいい場合があるでしょう。Twitter で開発者登録を行って、Twitter アプリの作成→ Firebase 管理画面上で登録を行って、Twitter ログインを実装してみてください。Facebook でも同じように登録が可能です。現状 Instagram や LINE ログインは公式にはありませんが、Functions をうまく使えば実現可能です。

ES6(ES2015) って？ Arrow Function ってなにそれおいしいの？

webpack の import 文についての部分でトランスコンパイラについて説明しました。その説明内の新しい書き方と表現していたものが **ES6** であり、古い書き方＝ブラウザ標準で動作する書き方が **ES5** です。

この ES というのは **ECMAScript** と呼ばれる JavaScript の標準であり、ES6 は 2015 年に標準化されたため ES2015 とも呼ばれます。

この場合の標準とは各社がそれぞれで作っているブラウザ上及びサーバー上で動作させる Javascript において「**こういう書き方をしたらどう動くか予め決めておく**」ことで、それによってプログラマが書きやすい新しい書き方等が、どの環境でも正しく同じに動くように整えられることに近づきます。

その中の一例として、ES6 で新しく標準化した Arrow Function(アロー関数) について説明します。今まで関数の定義は ES5 で

```
var fn = function (a, b) {
  return a + b;
};
```

と書いていたものが

```
var fn = (a, b) => {
  return a + b;
};
```

こう書いたり

```
var fn = (a, b) => a + b;
// 単一式の場合はブラケットや return を省略もできる
```

この様に記述したり出来るようになりました（関数内での this の扱いが変わる点だけ注意）。 他にもスコープド変数や Class 構文など、便利な機能が色々追加されています。

ES6 で追加されたシンタックスについてはひとまずこのあたりの記事を読んでおくと良

いと思います。
https://qiita.com/tuno-tky/items/74ca595a9232bcbcd727

一つ注意点として、Google Chrome などのブラウザでは ES6 で書いたコードもそのまま動作するため、babel などを挟んでいない環境でも ES6 で書いてしまった場合に IE などで動作しなくなる事もあるためご注意ください。IE を使わなければ大体全ては丸く収まりますが。。。

4.4. ログイン状態のチェック

次はサイト上でログイン状態をチェックします。App.vue 内で、チェックした内容は isLogin に格納します。

▼ /src/App.vue

```
created: function() {
  firebase.auth().onAuthStateChanged(user => {
    console.log(user);
    if (user) {
      this.isLogin = true;
    } else {
      this.isLogin = false;
    };
  });
},
```

data の後に created という関数を定義します。この関数は Vue.js がそのコンポーネントを作成したタイミングで実行されます。created の中で、firebase のログイン情報の更新がされたらまたその中の関数を実行し、ログイン状態であれば user という変数にユーザー情報が格納されるようになっています。「user が存在したらログインしている」として、isLogin に true/false を格納しましょう。

どうでしょうか、ログインした状態ではエディター画面が表示されましたでしょうか？ログアウトについてはまた先の章で行います。また、ログインした状態で console.log で user がブラウザコンソールに書き出されましたでしょうか？次の章ではそのデータを表示してみます。

コンポーネントのライフサイクルについて

先程利用した created は、コンポーネントが作成されたタイミングでしたが、他にも色々なタイミングで実行される関数があります。

https://jp.vuejs.org/v2/guide/instance.html
ここに載っている図がその全てですが、ざっくり説明すると、

beforeCreate・created：コンポーネントを作成する前後

beforeMount・mounted：コンポーネントを作成し、描画が終わる前後

beforeUpdate・updated：data 等が更新され描画内容を変更する前後

beforeDestroy・destoroyed：コンポーネントが破棄される前後

の計 4 つのイベントの前後で計 8 箇所あります。
描画終わったタイミングで実行したい内容であれば mounted、値が変更されるタイミングに実行したいなら、、、とその時々でベストな関数を選んでください。
他にもコンポーネント内部で、data で定義した変数に関して変更があったら実行する関数（watch）なども色々設定出来ますので、公式マニュアルを一通りさらっとでも読んでおくと、便利な機能を使いこなせるはずです。

4.5. コンポーネント間の情報の受け渡しとログイン情報の表示

次にログインユーザーの情報を画面に表示してみます。その際にログイン情報は App.vue が所有していますが、表示しているのは Editor.vue です。App.vue 内で表示してもいいですが、その後他のデータも利用するため、Editor.vue で取得したデータを表示出来るように値の受け渡しを行います。
そのために App.vue で新しい data を定義し、ユーザー情報を格納し、その後をのデータを Editor に引き渡すため次のように編集します。

▼ /src/App.vue

```
<template>
  <div id="app">
```

```
  <Home v-if="!isLogin"></Home>
  <Editor v-if="isLogin" :user="userData"></Editor>
</div>
</template>

<script>
import Home from './components/Home.vue';
import Editor from './components/Editor.vue';

export default {
  name: 'app',
  data() {
    return {
      isLogin: false,
      userData: null,
    }
  },
  created: function() {
    firebase.auth().onAuthStateChanged(user => {
      console.log(user);
      if (user) {
        this.isLogin = true;
        this.userData = user;
      } else {
        this.isLogin = false;
        this.userData = null;
      };
    });
  },
  components: {
    'Home': Home,
    'Editor': Editor,
  },
}
</script>
```

Firebase から取得したデータは user に格納されているので、それを userData に格納します。その後 Editor を読み出す際に :user=”userData” として、呼び出すコンポーネントにデータを引き渡すことが出来ます。

それを今度は Editor 側で取得するには下記にようにします。また、ついでにログアウトも出来るようにします。

▼ /src/components/Editor.vue

```
<template>
<div class="editor">
  <h1> エディター画面 </h1>
  <span>{{ user.displayName }}</span>
  <button @click="logout"> ログアウト </button>
</div>
</template>

<script>
export default {
  name: 'editor',
  props: ['user'],
  data() {
```

```
    return {}
  },
  methods: {
    logout: function() {
      firebase.auth().signOut();
    }
  }
}
</script>
```

props という名前で親コンポーネントから受け継ぐデータを定義します。firebase のユーザーデータには displayName というキーでユーザー名が格納されているので、template 内でそれを表示します。

ログアウトに関しても Firebase の signOut のメソッドを実行します。するとそれをまた App.vue が検知して表示を Home.vue に切り替えてくれます。

ちょい足しポイント

今回のログインでは App.vue でデータを管理していましたが、これがたくさんのコンポーネントでやり取りを行うようになると煩雑になりがちです。また、親から子へのデータの引き渡しはいいですが、例えばこの場合の Editor から Home という兄弟コンポーネントでのデータは受け渡せません。

そんな時には Vuex という公式ライブラリを利用し、Store を定義し、そこにデータを格納することで煩雑さを防ぐ手法が一般的です。

Store の定義の説明を行うと長くなってしまうため本書では割愛しましたが、規模が大きくなってくると必須な項目なので、是非 Vuex の公式ドキュメントを確認していただければと思います。

https://vuex.vuejs.org/ja/

また、今回のアプリケーションはログイン後に、ログインしているかどうかが確認できるまで Home.vue が表示されてしまっていると思います。これも firebase のログインチェックを行うまではローディング画面を出すなどすれば防ぐことが出来ますので、是非実装してみてください。

いかがでしたでしょうか。晴れて Google アカウントでログインし、ログインしたデータをアプリケーション内で表示することができました。ユーザーデータは他にもアカウントに設定している画像なども取得できますので、それを画面に表示するなど、ご自身で色々と試しつつカスタマイズしながら進めてもらえればと思います。

5. エディターの作成：データベース作成とデータ保存

さていよいよエディターを作って、メモを保存出来る機能の実装に移っていきます。ここではFirebase でのデータ保存や削除機能を使っていきます。

5.1. メモを編集出来るマークダウンエディターを作る

まずはブラウザ上でメモを書けるマークダウンエディターを作ります。textarea という html タグでは文章を書くことが可能なので、それとマークダウンエディターなのでマークダウンをプレビューする機能を作りましょう。

マークダウンをプレビューするには、そのためのライブラリを導入してみましょう。ターミナルに移り、プロジェクトのルートで以下のコマンドを実行します。

```
$ npm install --save-dev marked
```

こちらは makred というマークダウン記述を HTML 記述に変換してくれる npm モジュールです。

次に Editor.vue を編集します。

▼ src/components/Editor.vue

```
<template>
<div class="editor">
  <h1> エディター画面 </h1>
  <span>{{ user.displayName }}</span>
  <button @click="logout"> ログアウト </button>
  <div class="editorWrapper">
    <textarea class="markdown" v-model="markdown"></textarea>
    <div class="preview" v-html="preview()"></div>
  </div>
</div>
</template>

<script>
import marked from 'marked';
export default {
  name: 'editor',
  props: ['user'],
```

```
  data() {
    return {
     markdown: '',
    }
  },
  methods: {
    logout: function() {
      firebase.auth().signOut();
    },
   preview: function() {
    return marked(this.markdown);
   },
  }
}
</script>
<style lang="scss" scoped>
.editorWrapper{
 display: flex;
}
.markdown {
  width: 50%;
  height: 500px;
}
.preview {
   width: 50%;
   text-align: left;
}
</style>
```

こちらが無事に実行出来ていたら、左側の textarea に書き込んだマークダウンが内容が右側にリアルタイムにプレビューされていると思います。

5.1.1. script について

まず script 部分ですが、npm でインストールした marked を import で読み込んでいます。その後 data 関数ではオブジェクトに markdown というキーを追加しており、そこにマークダウン記述のテキストを入れることとします。

そのテキストをプレビューするために preview という関数を追加します。先程インストールした marked を利用して、markdown に格納されたテキストを HTML で返却します。

5.1.2. template について

次に template では、textarea を定義し v-model という属性へ data 関数で定義した markdown を入れます。v-model は input や textarea の状態をコンポーネントのデータへ格納するもので、この記述を加えるだけで textarea に書き込んだテキストが自動的に v-model で指定した変数へ格納されます (これを**データバインディング**と呼びます)。

その格納された値を利用し、preview では v-html という記述を利用します。これはその名前の通り、v-html で指定された値を直接 HTML として描画する機能になります。また、その指定するデータについて今回は methods の preview に () を付与して記述することで preview 関数の実行結果がデータとして入力されます。

※ v-html はブラウザでの XSS の原因となるため、テキストを共有するタイプのサイトで利用する際にはその対策を追加する必要が出てきます。詳細については 6.3 で記載します。

5.1.3. Style について

マークダウンプレビューが下に出ていては見づらいため並列になるような CSS を記述します。

ここでは表示の確認レベルのため最低限の記述とします。多分デザイナーの方はそろそろ見た目の無骨さに耐えられなくなってきている頃だと思います。ご自身で編集が出来る方は CSS をガシガシ書き変えながら進めていただければと思います。

そして今回記述した style タグに **scoped** の記述が追加されていることに気づいたと思います。実はこの記述を加えると、そのコンポーネントで記述した CSS は**そのコンポーネント内でしか適用されない**という便利機能が備わっています。

デベロッパーツールで template で記述した html について確認すると、data-v-1234567 のような属性がついているのが分かると思います。scoped の記述を行うと、自動的に html に個別の属性を割り当てその属性にのみ CSS が当たるように変換してくれるという機能になっています。

CSS はもともとページ全体の指定されたセレクタすべてに適用するという仕様のため、BEM 等のテクニックを用いてその影響範囲を絞って書くようなことがなされていました。しかし scoped という機能によって、テクニックが無くとも自身のコンポーネント以外での CSS の影響がなくなり、コンポーネント単位での作業に集中することが出来るようになりました。

5.2. メモを複数作成可能にする

せっかくなのでメモを複数作れるようにしたいですよね。この章では以下の機能を追加します。

- メモを保存する変数を配列に変更し、複数保存出来るようにする
- メモの一覧を作る
- メモの 1 行目を一覧で表示するタイトルとする
- 配列へメモを追加する
- メモを一覧から選択して切り替える（選択しているメモは色を変える）

これらをまとめて以下のように編集し、それぞれの実装について説明していきます。(各節で関わる部分のみ、同じコードを抽出したものを書きます。)

▼ /src/components/Editor.vue

```
<template>
<div class="editor">
  <h1> エディター画面 </h1>
  <span>{{ user.displayName }}</span>
  <button @click="logout"> ログアウト </button>
  <div>
    <div class="memoListWrapper">
      <div class="memoList" v-for="(memo, index) in memos" @click="selectMemo(index)" :data-
```

```
selected="index == selectedIndex">
          <p class="memoTitle">{{ displayTitle(memo.markdown) }}</p>
        </div>
        <button class="addMemoBtn" @click="addMemo"> メモの追加 </button>
      </div>
      <textarea class="markdown" v-model="memos[selectedIndex].markdown"></textarea>
      <div class="preview" v-html="preview()"></div>
    </div>
  </div>
  </template>

  <script>
  import marked from 'marked';

  export default {
    name: 'editor',
    props: ['user'],
    data() {
      return {
        memos: [{
        markdown: ''
        }],
        selectedIndex: 0
      }
    },
    methods: {
      logout: function() {
        firebase.auth().signOut();
      },
      addMemo: function() {
       this.memos.push({
        markdown: ' 無題のメモ ',
       })
      },
      selectMemo: function(index) {
       this.selectedIndex = index;
      },
       preview: function() {
         return marked(this.memos[this.selectedIndex].markdown);
       },
      displayTitle: function(text) {
       return text.split(/\n/)[0];
      },
    }
  }
  </script>
  <style lang="scss" scoped>
  .memoListWrapper {
     width: 19%;
     float: left;
     border-top: 1px solid #000;
  }
  .memoList {
     padding: 10px;
     box-sizing: border-box;
     text-align: left;
     border-bottom: 1px solid #000;
     &:nth-child(even) {
        background-color: #ccc;
     }
     &[data-selected="true"] {
        background-color: #ccf;
     }
  }
```

```
.memoTitle {
  height: 1.5em;
  margin: 0;
  white-space: nowrap;
  overflow: hidden;
}
.addMemoBtn {
  margin-top: 20px;
}
.markdown {
   float: left;
   width: 40%;
   height: 500px;
}
.preview {
   float: left;
   width: 40%;
   text-align: left;
}
</style>
```

5.2.1. メモを保存する変数を配列に変更し、複数保存出来るようにする

data 関数内でもともとメモが入っていた markdown という変数を memos という配列にし、中にオブジェクトを格納しておきます。このオブジェクト内のテキストデータのキーを markdown としておきます。

そしてその配列の中から、編集・プレビューしているデータは selectedIndex という変数で配列の番号を指定し表示することにしましょう。template の textarea と preview で指定しているデータもそれに合わせて変更しておきます。

```
<textarea class="markdown" v-model="memos[selectedIndex].markdown"></textarea>
〜〜〜
 data() {
   return {
    memos: [{
     markdown: ''
    }],
    selectedIndex: 0
   }
 },
```

5.2.2. メモの一覧を作る

memoList というクラスの要素でメモの一覧を表示します。その際 v-for という属性を付与することで配列やオブジェクトに応じた表示（リストレンダリング）を行います。v-for に配列かオブジェクトを指定しておくとデータの分その要素が生成されるという機能になります。

今回の記述では配列の中身のデータは memo に格納されており、配列の番号は index に格納されます。memokey 名で中のデータへアクセスが可能です。

```
  <div class="memoList" v-for="(memo, index) in memos" @click="selectMemo(index)" :data-selected="index == selectedIndex">
    <p class="memoTitle">{{ displayTitle(memo.markdown) }}</p>
  </div>
```

せんせ〜！ v-for と v-if を同じ要素につけるとうまく表示されませ〜ん

例えば配列の中で、データが入っていない要素は表示したくないという場面があると思います。そんなときにはつい

```
<p v-for="text in texts" v-if="text">{{ text }}</p>
```

の様な形で記載しがちです。しかし v-for と v-if は同じ要素に付与することは出来ないため、v-if の中身にかかわらず表示されてしまいます。そんなときには、以下の様にその要素を囲む要素を作るとよいでしょう。

```
<template v-for="text in texts">
  <p v-if="text">{{ text }}</p>
</template>
```

これならリストレンダリングを使いつつ、要素の表示非表示を制御出来ます。

template 要素とは、こういったリストレンダリングや条件付きレンダリングの際などに利用するための要素で、html として意味を持たせない場合などに利用するとよいでしょう。

または「テキストが空の時には非表示」というのが決まっている場合は

```
<p class="hoge" v-for="text in texts">{{ text }}</p>
```

で、CSS で

```
.hoge:empty{
  display:none;
}
```

のように書いてもいいかもしれません。v-if の条件に応じて使い分けてみてください。

5.2.3. メモの 1 行目を一覧で表示するタイトルとする

一覧に表示するメモのタイトルは今回はメモの 1 行目とします。displayTitle という method を追加し、そこでは入力されたテキストデータの 1 行目を返却するようにします。

split(/\n/) ではテキストを改行で分割し配列にします。その配列の初めの値を返却することで 1 行目だけになります。また、文字数が多い場合は CSS で要素からはみ出た文は表示しないようにします。

```
  displayTitle: function(text) {
    return text.split(/\n/)[0];
  },

~~~

.memoTitle {
  height: 1.5em;
  margin: 0;
  white-space: nowrap;
  overflow: hidden;
}
```

5.2.4. 配列へメモを追加する

メモを追加ボタンをメモ一覧の最後に配置し、押されたら配列にデータが追加される method を呼び出します。

```
<button class="addMemoBtn" @click="addMemo"> メモの追加 </button>
~~~
  addMemo: function() {
    this.memos.push({
      markdown: ' 無題のメモ ',
    })
  },
```

5.2.5. メモの一覧を選択して切り替える

@click でそのメモを選択して表示を切り替えるために selectMemo を method へ追加します。その際 index を入力し selectIndex を切り替えることで表示・プレビューともに切り替わるようにします。

```
  <div class="memoList" v-for="(memo, index) in memos" @click="selectMemo(index)" :data-
selected="index == selectedIndex">
  ~~~
    selectMemo: function(index) {
      this.selectedIndex = index;
    },
```

5.2.6. 選択しているメモは色を変える

どのメモを選択しているかわからなくなるため、選択している要素は data-selected="true" という属性がつくようにします。色は CSS で指定します。

属性について、データに応じて内容の変更がある場合には明示的に記述しておく必要があります。Vue.js では ":" を属性の頭に付与することで、今回ではメモの index が現在選択されているものと一致した場合に属性が付与されるようになります。: は v-bind: の略称記法で、どちらでも書いても OK です。

```
  <div class="memoList" v-for="(memo, index) in memos" @click="selectMemo(index)" :data-
selected="index == selectedIndex">
  ~~~
  .memoList
    &[data-selected="true"] {
      background-color: #ccf;
    }
  }
```

ちょい足しポイント

「メモの順番を変えられる機能」「メモ毎に最後に編集した日付や作成した日付を追加」などをしてみてもいいと思います。今回配列にテキストだけでなくオブジェクトとして追加したのはそういった要素を追加しやすいように考えての事でした。

markdown 以外に、タグだったり、プレビューの ON,OFF 設定等自由に情報を追加してみてください。

さらに高度な機能としては

・検索機能

・ソート機能

などでしょうか。lodash(https://lodash.com/docs/) などのライブラリを使うと結構あっさり実装出来たりもするので、是非挑戦してみてください。

どうでしょうか。メモが複数編集でき、リアルタイムプレビューもされることでマークダウンエディターっぽさが出てきたと思います。

今のままではアクセスしてメモを書いてもリロードすると消えてしまいます。次の章からはいよいよメモを Firebase へ保存することでより実用的なものにしていこうと思います。

5.3. メモの削除機能追加

メモを追加できたので、次は削除機能を追加します。
まずは template に削除ボタンを追加します。

▼ /src/components/Editor.vue

```
    <button class="addMemoBtn" @click="addMemo"> メモの追加 </button>
    <button class="deleteMemoBtn" v-if="memos.length > 1" @click="deleteMemo"> 選択中のメモの削除 </button>
    </div>
```

次に対応する deleteMemo 関数を methods 内に追加します。

▼ /src/components/Editor.vue

```
methods に追加
deleteMemo: function() {
  this.memos.splice(this.selectedIndex, 1);
  if (this.selectedIndex > 0) {
    this.selectedIndex--;
  }
},
```

splice は配列の任意の位置からデータを取り出す関数です。

ちょい足しポイント

取り出したデータをゴミ箱として保存し、振り返れるようにしてみて下さい。splice は返り値で取り出したデータが取得出来ます。

5.4. Firebase Realtime DB の rule を設定する

Firebase RealtimeDB のドキュメントは以下にありますので、ざっと目を通しておくと良いと思います。

https://firebase.google.com/docs/database/security

今回はソース内の memos の配列がユーザー毎に保存出来て読み書きが出来れば目的の動作に

なります。Realtime DB では、DB のルールを JSON 形式で設定出来ます。

「管理画面 → Database →（はじめるボタン）→ ルールタブ」

を開いて下さい。ここではルールの設定や、編集中のルールのエミュレーションが出来ます。

自分のメモが保存でき、そのメモが他のユーザーも読み込みが出来ないようにするため、マニュアルの以下の項目のサンプルにある ” ユーザー ” の項目を参考にします。

https://firebase.google.com/docs/database/security/quickstart?authuser=0#sample-rules

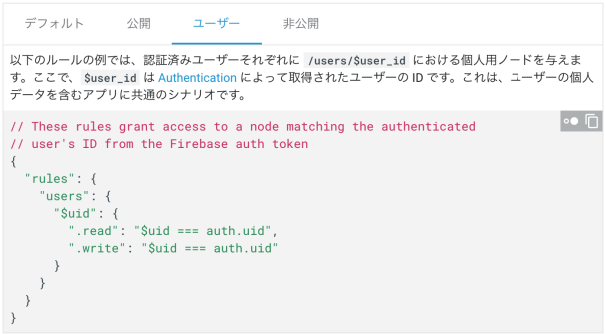


図 データベースルールのサンプル

rules 配下にルールを記載します。このサンプルの users は DB 内のパスで、$uid は user 配下にユーザー ID と同じ名前のパスへデータを配置するという意味になります。

このユーザー ID は firebase でログインしたユーザーに個別に振られる Firebase ログインのユーザー ID であり、App.vue 内で取得している user オブジェクト内に uid というキーで取得出来る “5fJlwVxnnhYKpMx65IFxREXfZal1” のような文字列が ID です。

他のユーザーが自分のデータを読み書きできないように設定するには、サンプル内の $uid === auth.uid がポイントで、”.read” は読み込み、”.write” は書き込みについてそれぞれ、**ユーザー ID と同じ名前のパス内のデータと、ログインしているユーザー (auth) のユーザー ID(uid) が一致する場合のみ可能**という設定になります。

今回はメモデータなので、memos というパスでデータを保存しようと思いますので、サンプルをコピーし、以下のように編集して保存しておきましょう。

```
{
  "rules": {
    "memos": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

図 データベースルールの設定

これで一応データベースへデータを保存する準備は OK です。次の章では編集したメモの保存をやってみましょう。

ちょい足しポイント

現状のルールでは指定された場所以外に、ユーザーが無理やりデータを保存（ログイン状態で自らプログラムで API を叩くような形）をしようとすれば出来てしまいます。そういったデータを追加されても他のユーザーに影響はないですが、指定のパス以外にデータが保存出来ないようなルールを追加してみましょう。

5.5. メモの保存と読み込み機能の作成

ここではデータベースへ保存する機能と、ページロード時にそれを読み込む機能を追加します。まずは保存するボタンを追加し、RealtimeDB へ保存する処理を書きます。

▼ /src/components/Editor.vue

```
  <button class="deleteMemoBtn" v-if="memos.length > 1" @click="deleteMemo">選択中のメモの削除</
button>
      <button class="saveMemosBtn" @click="saveMemos">メモの保存</button>
    </div>
~~~~
script:methods 内関数追加
  saveMemos: function() {
    firebase
      .database()
      .ref('memos/' + this.user.uid)
      .set(this.memos);
  },
~~~~
style 内追加
deleteMemoBtn {
  margin: 10px;
}
```

以前ログインの際にも利用した firebase に対して、.database() を実行しデータベースに接続し、.ref(‘/memos’ + this.user.uid) ではデータを読書するパスを指定します。ルール設定を行ったパスですね。

そして .set(this.memos) でそのパスへデータを保存することが出来ます。set はそのパス以下のデータを指定したデータに上書きするため、例えば update に変えると、指定したデータ以外

の部分のみ更新するような機能を備えています。

これで保存が出来たので、次にページ読み込み時にデータの読み込みを実行するようにしましょう。

▼ /src/components/Editor.vue

data 関数の次に追加

```
created: function() {
  firebase
   .database()
   .ref('memos/' + this.user.uid)
   .once('value')
   .then(result => {
    if (result.val()) {
      this.memos = result.val();
    }
   })
 },
```

created 関数内に読み込みの処理を追加します。

書き込み時と同じように database への接続、パスの指定を行い読み込みの場合は .once('value') を指定します。この once は一回だけの読み込みに利用します。

realtimeDB では元データに変更があった際に通知してくれる機能があり、詳細は以下のドキュメントに記載されています。

https://firebase.google.com/docs/database/web/read-and-write?authuser=0#listen_for_value_events

その後データを受け取るには、.then(result =>{}) という Promise 形式でデータを受信出来ます。この例ではデータの読み込みが終わり次第 then 内に定義した関数が結果とともに読み込まれるという流れです。

この result で .val() というメソッドを実行することで、指定のパスのデータを取得するこができます。初めて利用するユーザーの場合は結果が null なので、if 文を追加しデータがあった場合のみ memos を上書きするという処理とします。

ちょい足しポイント

いちいち保存ボタンを押すのは面倒なので

・Ctrl + s を押下で保存

・一定時間経ったら自動的に保存

・編集の区切り（メモを切り替えたら）保存

などの機能を追加するともっと使いやすくなりますね。

これで一通りの機能の開発は終了です。お疲れ様でした。見た目はどうであれ、これで無事にオリジナルの「マークダウンをオンライン上で保存し編集出来る Web サービスの開発」が出来ました。是非デプロイも試してみてください。

次の章からは作ったサービスを公開するまでに必要な項目をなぞっていきます。Vue.js・Firebase で開発する項目についてもここで終了になるため、それらの習得のみが目的の方は次の章は読み飛ばして頂いても構いません。

6. Web サービスとして公開するまでの必要な準備

さて開発もいよいよラストスパート！この章にある内容を概ね網羅して頂いて、より良い Web サービスを目指していってもらえればと思います。

6.1. 見た目を整える

今までの章では機能面での開発にフォーカスしていたので、見た目は気にしていませんでした。この章ではデザイン面で考慮しておきたい実装ポイントを紹介していきます。

ロゴの作成やフォントの選定、カラーリング等に関しては触れません。

6.1.1. リセット CSS を導入する

ブラウザで標準で入っている CSS をリセットすることが目的の CSS モジュールです。標準で入っている CSS は margin 等が入っているため意図通りのデザインにならない原因となることが多いため、一度それらを消してから CSS を書いていくためです。

reset.css や、normalize.css、shitaji.css などなど、微妙に違いますが目的は一緒のモジュールがいくつかあります。これらはすべて npm で公開されているので、npm install した後に以下の様に編集すればすぐ利用することが出来ます。

今回は shitaji.css を利用してみたいと思います。

▼ /src/main.js

```
import 'shitajicss/docs/css/shitaji.min.css';
import Vue from 'vue'
~~~
```

webpack では CSS も JS と同じく一緒にまとめてくれる機能があるため、このように Javascript のライブラリの様に CSS ファイルも import すれば利用することが出来ます。

また、npm によっては読み込みたいファイルの場所にバラツキがあることもあります。そのときには install 後に node_modules の中のパッケージディレクトリの中を見て目的のファイルを探しましょう。

もちろんリセット CSS 自体量は少ないのでオリジナルで作っても良いと思います。今回のプロジェクトではマークダウンのプレビュー部分もリセットされたままだとすべて平文になってしまうため、プレビュー部分には追加で見やすくなるような CSS を書いてみましょう。

6.1.2. CSS ファイルの管理

前章で CSS(SCSS) ファイルを import 出来ることがわかっていただけたと思います。こちらはライブラリ以外にも自分で作ったファイルを読み込ませることももちろん可能です。

5.1.3. では scoped CSS についての説明を書きました。好みもアリますが、基本的に component 内 CSS はすべて scoped にしてしまい、全体で使うモジュール等は別ファイルとして切り出しておくと、見通しが付きやすいと思います。人によっては全体で利用する CSS は App.vue のなかで scoped にせずにそのまま書くという方もいると思いますのでお好みで実装してください。

具体的には src ディレクトリに scss ディレクトリを作成し、サイト全体で利用する CSS を書いていきましょう。style.scss ファイルを作成し、App.vue に記載されていた style は style.scss ファイルに移動してみましょう。その後、その style.scss ファイルは main.js で読み込みを行います。

▼ /src/main.js

```
import 'shitajicss/docs/css/shitaji.min.css';
import './scss/style.scss';
import Vue from 'vue'
~~~
```

これで、全体で利用する CSS は style.scss へ、コンポーネント毎の CSS はコンポーネント内に書くという切り分けが出来たと思います。

6.2. トップページにサービスの説明文を加えよう

たとえどんなに素晴らしいサービスだったとしても使われなければ意味がありません。そのためにも、TOP ページに来たユーザーにできるだけ登録してもらえるように工夫しましょう。

これはどんなサービスで、何が出来て、どんな素晴らしい体験が得られるのかをしっかりと説明して、スクリーンショットやサンドボックスなどを入れてサービスの魅力をできるだけ伝えましょう。今回は Google アカウントを利用しているので、登録手続き自体はすぐに終わるのでその点をアピールしてもいいかもしれません。

様々な Web サービスを見てどんなトップページなら登録したくなるか、というのを研究して是非ご自身が登録したくなるようなページを作ってみて下さい。

6.3. 利用規約・プライバシーポリシーを記載する

公開してユーザーに利用してもらうタイミングで利用規約と、今回はユーザー登録してもらうためプライバシーポリシーも作成しておきましょう。

こちらのサイトにサンプル文言と、どういったことを書く必要があるかが詳細に書かれているため本書では内容を省きます。

Web サイトの利用規約 : http://kiyaku.jp/

このサイトのものを参考にして頂いて、更に類似サービスのものを一通り舐めて、足りない点があれば随時追記するとい形で文章を作っていけばいいと思います。
文章が出来たら HTML として作ってサイトにリンクを設置するか、β版などの段階では Google docs に保存して誰でも参照可能な状態にしておくのも編集が楽なのでおすすめです。

6.4. XSS 対策などの最低限のセキュリティ対策

個人情報を扱う Web サービスを公開する上で怖いものの一つは情報漏洩ですね。ひとまず本書の通りに開発ができていれば現状特に心配することはありませんが、セキュリティ周りについて確認すべき事を書いておきます。

3.2 で記載したように、Firebase ではサイトにアクセスするデータの ID 等をユーザーから見える場所に配置しますが、これ自体は特に問題ありません。
ログインに関しては指定された URL 上でのみ許可を行うため、他のサイトでは動作しないようになっております。また、DB へのアクセスについては認証しなければアクセスできない＆認証し自分のデータのみしかアクセスできないというルールにしてあるため、他人に自分のメモが見られてしまうことはありません。
また、Firebase ではユーザーの認証後にアクセスできる情報（メールアドレスや名前等）については DB とは隔離してあり、たとえ開発者が DB のルールを失敗して設定していたとしても閲覧することは出来ないようになっています。
しかし自分のアカウントでログイン状態でそのデータ取得は出来るため、それをそのまま DB に保存すること自体は出来てしまいます。
もしシステム上必要で、そのようなデータを保存したい場合はルールをしっかり設定してそのユーザーのみが read 出来るように最低限のデータのみ保存してください。ルールの設定ミス等で DB の読み取りが他人から出来てしまうと、ユーザーデータの流出につながってしまいます。

本書ではマークダウンのプレビューに v-html で HTML 出力という機能を利用しています。この機能は誤った使い方をすると XSS(Cross-site scripting) という脆弱性を引き起こす可能性がある機能です。
例えば「自分のメモを他の人も閲覧可能な状態にする」という機能を追加で実装した場合に script タグを書いてそれを実行出来てしまうと、メモ内に「自分のログイン情報を送信する」という様なコードを書いて " メモを見た人の情報を盗む " という操作をやろうと思えば出来てしま

うような事になってしまう可能性があります。その為ユーザー間共有のような機能を今後開発するような場合には XSS が起こらないような対策の導入が必要です。

6.5. β版テストを行い、公開する

お疲れ様でした。もろもろ公開までの準備が終わりましたでしょうか？

公開前にやっておくと良いことの一つに、誰かにベータ版として使ってみてもらうことです。友人何人かに事前に試してもらいインタビューを行って、機能の説明が必要になる場面にはサイト上に補足を加えるなどしましょう。

また、公開前には Google Analytics を導入して PV などを計測すると後々データが見れるため導入すると良いと思います。

是非作ったサービスを公開するようになったら SNS で発信してみてください！ Twitter で @ nabettu 宛にリプライを送っていただければ私が使ってみます。

7. 最後に

7.1. フィードバック・ご意見・ご感想

初心者の方でもつまづかずに進められるように書いていますが、もしわからないことがあったり、全く同じように書いても動かない、誤字脱字やご意見ご感想ありましたら、以下のどれでも結構ですので、ご連絡いただければと思います。

Twitter: https://twitter.com/nabettu
Mail: t@cremo.tokyo
Facebook: https://www.facebook.com/nabettu
匿名フォーム : http://bit.ly/2EGH3uD

7.2. 筆者紹介

本文 / 渡邊 達明 (@nabettu)
Web サービスを作るのが生きがいのフロントエンドエンジニア。ブロガーの妻のスーパーヒモニート。

7.3. Special Thanks !

デザイン相談　たかゆり (@mazenda_mojya)
レビュアー　のびーさん (@fnobi)
レビュアー　姫ちゃん (@beeeeinto)
テスター　　しの (@shanonim)
色々な助言本当に助かりました！いつもありがとうございます。

7.4. 強くてニューゲーム

本書の内容では物足りない方のために " ちょい足し " の内容を各所に散りばめましたが、ちょい足しの内容をすべて追加した上で、まだまだ色々やってみたいぞ！というやる気に満ち溢れたフレンズのみんなは以下の内容に取り組んでみてください。

きっと終わるころには大体の Web サービスやその原型・プロトタイプが作れるようになっているはずです！

本書の内容に加えて私が作ったサンプルに追加した機能には★マークを付けております。github リポジトリの /feature/add-design ブランチに追加機能を入れたソースコードを見れるよ

うにしておきますのでもし気になった方は参考にしてみてください。

7.3.1. 初心者向け機能

★ナビゲーションバーの追加
★マークダウンでチェックボックスを表示出来るようにする
★ログイン時に表示まで Loading を入れる
★メモ削除時に確認する
★セーブ中はセーブボタンがローディングする（ローディング中は押せない）
開閉できるメニューを作ってみる
スマホと PC で別々な CSS を当てるようにする
保存前にタブを消そうとしたら警告を出す
メモの文字数を表示する
functions を使って API を作り、登録ユーザー数をトップページにだしてみる

7.3.2. 中級者向け

★ textarea をスクロールするととプレビューのスクロールも連動
★各種機能にショートカットキーの追加
変更したログをとっておいて、以前のメモへ戻れる
Functions を使ってサーバーサイドレンダリングをやってみる (XSS に注意)
Firebase RealtimeDB ではリアルタイムでの複数人編集機能があるので、それをとりいれて編集できるようにする
メモを public なデータとして、他の人が見えるが編集できないような場所におけるようにする。（XSS に注意）
画像を保存できるようにする。(Firebase には画像などのストレージもあります)
JSON でのエクスポート機能
外部連携（例えば evernote に保存出来るようにする、Twitter ログインして Functions で何かしらを API 経由でつぶやいてみる等）

他にも evernote などや Dropbox paper を参考にしつつ、痒いところに手が届く用な機能を考えて作ってみてください！

7.5. あとがき

最後まで読んでいただきありがとうございます！
本書は私自身が 2 年前にいわゆる Web 系の会社に業務未経験で入社した際に、右も左もわか

らなった私に先輩社員達から色々と教えてもらいなんとか業務をこなせる様になっていった経験を踏まえて「その頃の自分が今未経験入社したら何が知りたいか」というテーマを元に執筆してみました。

「jQuery を使い HTML と CSS でなんとなく Web サイトを作ってみたことはあるけど、次になに覚えたらいいんだろう？」
「SPA ってやつ使ってみたいけどハードル高いな〜」
「Web サービス作りたいんだけど Ruby on Rails は 3 回挫折してる」

本書がこのような声を上げる方にとってのとっかかりになる本になっていれば幸いです。
それではみなさん良きインターネッツライフを。

2018 年 4 月
渡邊 達明

初めてのシングルページアプリケーション Vue.js と Firebase で作るミニ Web サービス

2018 年 4 月 22 日 技術書典 4
著 渡邊達明 @nabettu