

SPA Firebase



Firestoreの概要

Firestoreの概要



BaaSやmBaasと呼ばれる
(mobile Backend as a service)

サーバー側で必要な仕組みを用意
サーバーを用意しないで開発ができるので
サーバーレスとも呼ばれる

Firebaseの歴史

2011年 Firebase Inc. が開発

2013年 Zapierを統合(複数のアプリ間通信)

2014年 Googleが買収

2017年10月 Cloud Firestore提供開始

2021年8月 Firebase JS SDK v9.0.0)

(他に iOS, Android, C++, Unityなども対応)

Firebaseの機能

Hosting ・ ・ HTML/CSS/JSが動く
レンタルサーバー CLIも用意

Cloud Firestore ・ ・ データベース(NoSQL)

Cloud Storage ・ ・ 画像保存など

認証(Authentication) ・ ・ SNS認証も

Cloud Functions ・ ・ Firebase上で
プログラムを動かす仕組み

パフォーマンス監視

グーグルアナリティクス

エクステンション ・ ・ 画像リサイズ、翻訳etc…

Firestore 無料プラン

Sparkプラン

Authentication 1万/月

Cloud Firestore 保存済みデータ合計 1 GiB

書き込み 2万件/日

読み取り 5万件/日

Hosting ストレージ 10GB

Cloud Storage GB保存済み 5GB

<https://firebase.google.com/pricing?hl=ja>



プロジェクト登録

Firestoreにログイン

2014年より

Googleのサービスになっていることもあり
Googleアカウントでログインできます。

(事前にGoogleアカウントを作成しておいて
ください。)

Firestoreプロジェクト登録

コンソールに移動
->プロジェクトを作成

プロジェクト名 nuxt-spa-book-app
アナリティクス(今回は無効)

ウェブアプリを追加

アプリにFirebaseを追加
ウェブのボタンをクリック
->アプリのニックネーム nuxt-spa

Firebase Hostingを設定する

Firebase SDK(開発キット)は後ほど



フォルダコピー

フォルダコピー



今後、
SPA、SSR、SSG、PWAなど
それぞれフォルダを分けて設定したい。
section3をベースに
Section4フォルダをつくり、
必要ファイルをコピーしておきます。

隠しファイルの表示方法

Mac

Cmd+Shift+.


Win

エクスプローラー->表示->隠しファイルに
チェック

コピー&インストール

.nuxt フォルダと
node_modules フォルダは
サイズが大きく、
npm install やビルドで生成できるので
コピー不要

npm ci でインストール
package-lock.json を参照し
依存パッケージをダウンロードする



firebase SDKの インストール・設定

firebase SDKのインストール

```
npm install firebase@"9.*"
```

ver8とver9で大きく変更

ver9は必要な機能をimportする形式
(軽量化&高速化)

公式もver9利用を推奨

※@nuxtjs/firebaseは将来対応？

plugins/firebase.js

```
// 初期化用関数をインポート
import { initializeApp } from 'firebase/app'

// コンフィグ設定
const firebaseConfig = { 略 }

// 初期化(インスタンス化)
const firebaseApp = initializeApp(firebaseConfig)

// 他ファイルで使えるようexport (this.$firebaseで使えるようになる)
export default ( context, inject ) => {
  inject('firebase', firebaseApp)
}
```


nuxt.config.js

```
export default {
```

略

```
  plugins: [
```

```
    '@plugins/firebase.js'
```

```
  ]
```

```
}
```




Cloud Firestore

Cloud Firestore

管理画面で有効化
テストモード

ロケーション(後で変更できない)

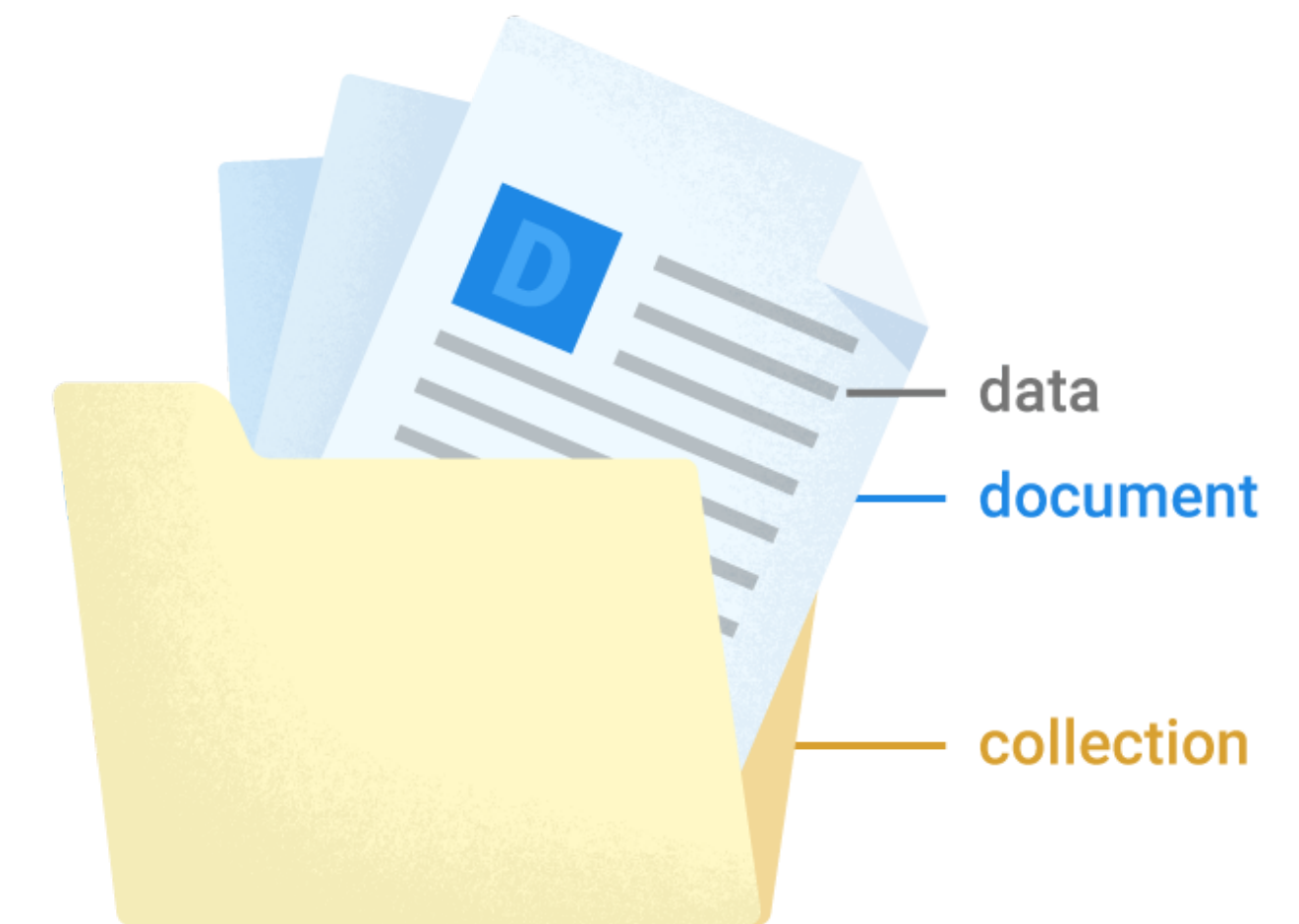
asia-northeast1 東京

asia-northeast2 大阪



Cloud Firestoreの用語

コレクション・・・フォルダ
ドキュメント・・・資料
データ・・・1つ1つのデータ



サブコレクションなどもある

firestoreの接続



getFirestoreの引数に
Firebaseインスタンスを渡せば
使う事ができる

```
import { getFirestore } from 'firebase/firestore'
```

```
const db = getFirestore(this.$firebase) // 接続
```


データの追加 1

pages/firebasetest/addData.vue

```
<template>
```

```
  <div>
```

```
    <v-btn @click="addData">追加</v-btn>
```

```
  </div>
```

```
</template>
```

```
<script>
```

```
// 必要な機能(関数)をインポート
```

```
import { getFirestore, collection, addDoc } from 'firebase/
firestore'
```

```
data(){ return { id: '001', title: 'テスト' }}
```

```
methods: { addData(){} }
```


データの追加 2

pages/firebasetest/addData.vue

// クラウド上にあり通信が必要なため

// 非同期関数(async/await) & try-catch構文

```
methods: {
```

```
  async addData(){
```

```
    try {
```

```
      const db = getFirestore(this.$firebase) // 接続
```

```
      const docRef = await addDoc(collection(db, 'tasks'), {  
        id: this.id,
```

```
        title: this.title
```

```
    })
```

```
    console.log('追加したデータのid:', docRef.id)
```

```
  } catch (e){ console.error('error:', e) } }
```

```
}
```


データの取得 1

pages/firebasetest/showData.vue

```
<template>
  <div>
    <div v-for="(task, index) in tasks" :key="index">
      id: {{ task.id }} / title: {{ task.title }}
    </div>
  </div>
</template>
<script>
import { getFirestore, collection, getDocs } from 'firebase/firestore'
export default {
  data(){ return { tasks: [] } },

  async created(){}
}
```


データの取得 2

pages/firebasetest/showData.vue

// クラウド上にあり通信が必要なため

// 非同期関数(async/await) & try-catch構文

```
async created(){
```

```
  try{
```

```
    const db = getFirestore(this.$firebase)
```

```
    const querySnapshot = await getDocs(collection(db, 'tasks'))
```

```
    querySnapshot.forEach( doc => {
```

```
      this.tasks.push(doc.data()) // 配列に追加
```

```
      console.log(doc.id, doc.data()) //doc.dataがそれぞれのデータ
```

```
      console.log(doc.data().id)
```

```
      console.log(doc.data().title) })
```

```
  } catch(e){ console.error('error:', e)  } }
```

```
}
```


firestoreのマニュアル



firebase web

<https://firebase.google.com/docs/web/setup?authuser=0>

構築->firestore

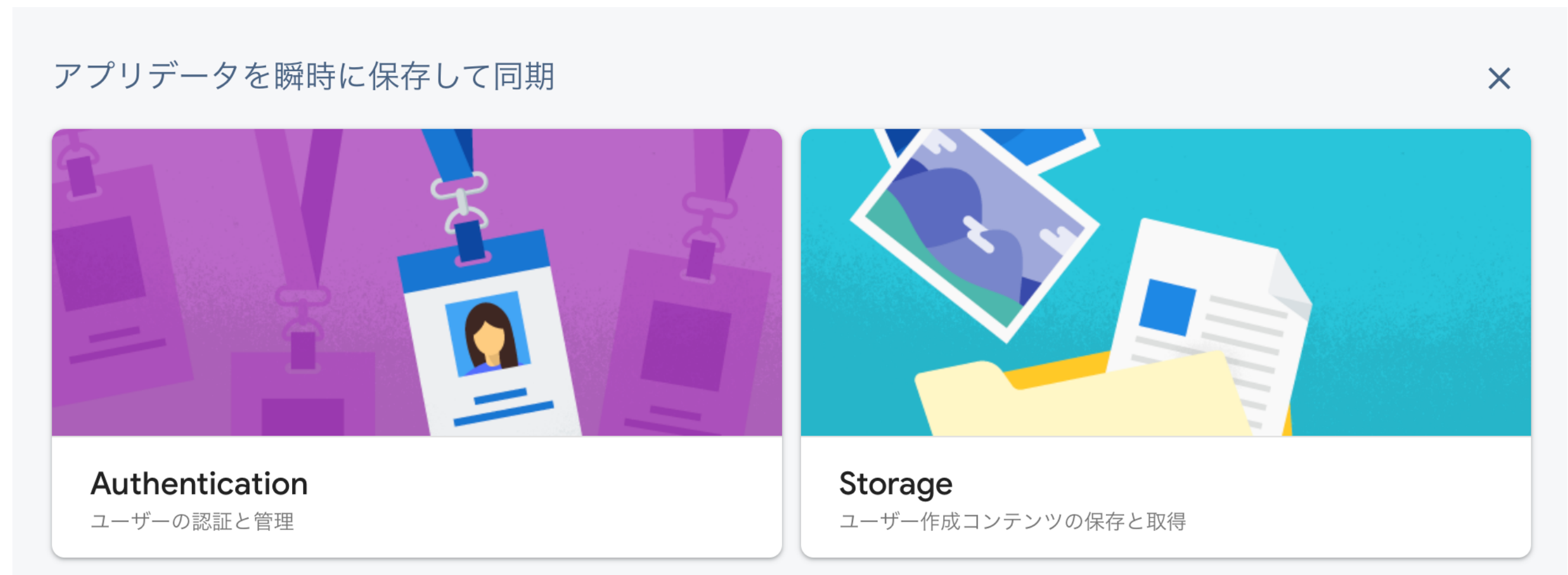
<https://firebase.google.com/docs/firestore?authuser=0>



Authentication

認証・管理

Authentication



管理画面で有効化
メール・パスワード
追加のプロバイダ(Google,
Twitter, Facebook等)

認証・・・できる事

機能	メソッド
認証機能の利用	getAuth
ユーザー登録 (メール&パスワード)	createUserWithEmailAndPassword
サインイン	signInWithEmailAndPassword
認証状態の変更	onAuthStateChanged
認証状態の永続性	setPersistence

ユーザー登録 1

pages/auth/register.vue 抜粋

```
<template>
```

```
<div>
```

```
<v-text-field v-model="email">メールアドレス</v-text-field>
```

```
<v-text-field v-model="password">パスワード</v-text-field>
```

```
<v-btn @click="signUp">ユーザー登録</v-btn>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
export default {
```

```
  data(){ return { email: "", password: "" } }
```

```
  methods: {
```

```
    signUp(){}
```

```
}
```

```
</script>
```

ユーザー登録 2

pages/auth/register.vue 抜粋

```
<script>
```

```
import { getAuth, createUserWithEmailAndPassword } from 'firebase/auth'
```

```
// 略
```

```
signUp(){
```

```
  const auth = getAuth(this.$firebase)
```

```
  createUserWithEmailAndPassword(auth, this.email, this.password)
```

```
  .then( userCredential =>{
```

```
    console.log( userCredential.user )
```

```
    console.log('ユーザー登録ok!')
```

```
  }).catch( e => {
```

```
    alert (e.message )
```

```
    console.error('error:', e)
```

```
  })
```

```
}
```

```
}
```

```
</script>
```


ログイン 1

pages/auth/login.vue 抜粋

```
<template>
```

```
  <div>
```

```
    <v-text-field v-model="email">メールアドレス</v-text-field>
```

```
    <v-text-field v-model="password">パスワード</v-text-field>
```

```
    <v-btn @click="login">ログイン</v-btn>
```

```
  </div>
```

```
</template>
```

```
<script>
```

```
export default {
```

```
  data(){ return { email: "", password: "" } }
```

```
  methods: {
```

```
    login(){}
```

```
  }
```

```
</script>
```

ログイン 2

pages/auth/login.vue 抜粋

```
<script>
import { getAuth, signInWithEmailAndPassword } from 'firebase/auth'
// 略
login(){
  const auth = getAuth(this.$firebase)
  signInWithEmailAndPassword(auth, this.email, this.password)
    .then( userCredential =>{
      console.log( userCredential)
      console.log('ログインok!')
    }).catch( e => {
      alert (e.message )
      console.error('error:', e)
    })
}
}
}
</script>
```


ログイン中か確認が必要

ページ表示時

ログインしているかの確認

->ログインしていればそのまま表示

->していなければログインページにリダイレクト

ログイン必要な全てのページで確認が必要

ログイン中か確認するために

firebase/auth ・ ・ firebaseの機能

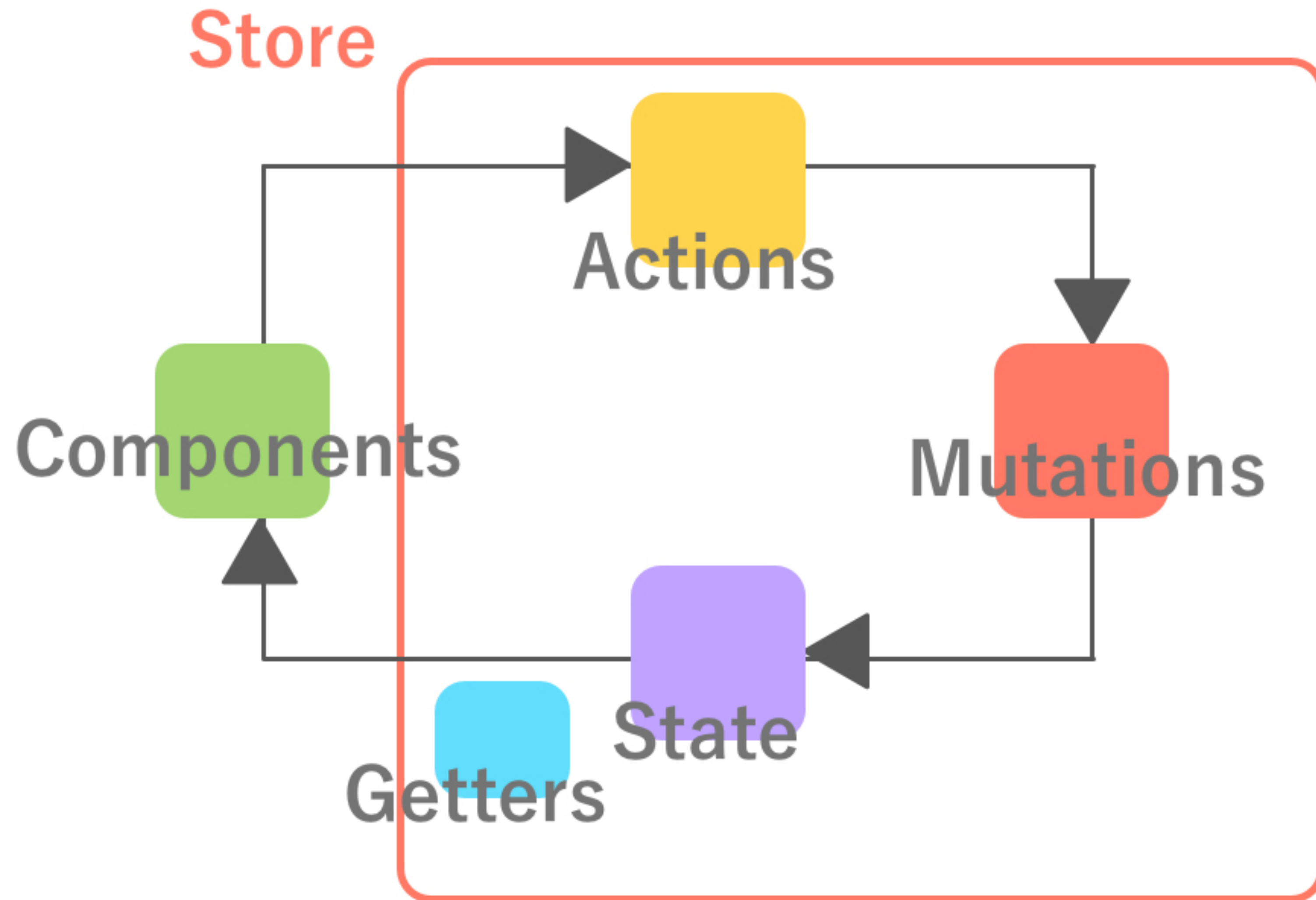
middleware ・ ・ 読込時に認証状態の確認

vuex ・ ・ 認証状態の保持
グローバル変数のように
どのページからでも参照可能



Vuex

Vuex 簡易図



Vuex OptionsAPIとの対応表

Options API	Vuex	Memo
data	state	
computed(get)	getters	プロパティなら キャッシュあり
methods	mutations	同期 履歴残る
methods	actions	非同期

Vuex 使い方・引数

Vuex	呼び出し	引数
state	<code>\$store.state.xxx</code>	
getters	<code>\$store.getters.xxx</code> <code>\$store.getters('xxx')</code>	state, [getters]
mutations	<code>\$store.commit('xxx')</code>	state, {値(payload)}
actions	<code>\$store.dispatch('xxx')</code>	context, {値(payload)} ※{commit} {state}など含む

Vuex サンプル1-1

<https://nuxtjs.org/ja/docs/directory-structure/store>

store/index.js

```
export const state = () => ({  
  counter: 0  
})
```

```
export const mutations = {  
  increment(state){  
    state.counter++  
  }  
}
```

Vuex サンプル1-2

pages/vuextest.vue

```
<template>
  <div>
    {{ $store.state.counter }}
    <v-btn @click="$store.commit('increment')">+</v-btn>
  </div>
</template>
```


Vuex サンプル2-1

store/todos.js

```
export const state = () => ({ list: [] })
```

```
export const mutations = {  
  add(state, text) {  
    state.list.push({  
      text,  
      done: false  
    })  
  },  
  remove(state, { todo }) {  
    state.list.splice(state.list.indexOf(todo), 1)  
  },  
  toggle(state, todo) {  
    todo.done = !todo.done } }  
}
```

Vuex サンプル2-2

pages/todos.vue

```
<template>
  <ul>
    <li v-for="todo in todos" :key="todo.text">
      <input :checked="todo.done" @change="toggle(todo)" type="checkbox">
      <span :class="{ done: todo.done }">{{ todo.text }}</span>
      <input @change="remove(todo)" type="checkbox">
    </li>
    <li><input @keyup.enter="addTodo" placeholder="What needs to be
done?"></li>
  </ul>
</template>
```


Vuex サンプル2-3

pages/todos.vue

```
<script>
```

```
import { mapMutations } from 'vuex'
```

```
export default {
```

```
  computed: {
```

```
    todos () {
```

```
      return this.$store.state.todos.list } }, // store/index.js以外はファイル名も指定
```

```
  methods: {
```

```
    addTodo (e) {
```

```
      this.$store.commit('todos/add', e.target.value) // ファイル名/メソッド名
```

```
      e.target.value = ''
```

```
    },
```

```
    ...mapMutations({
```

```
      toggle: 'todos/toggle', // ファイル名/メソッド名
```

```
      remove: 'todos/remove'
```

```
    }) } }
```

```
</script>
```

```
<style>
```

```
.done { text-decoration: line-through; }
```

```
</style>
```




Authentication & Vuex

store/auth.js

```
export const state = () => ({})
```

```
export const mutations = {}
```

```
export const actions = {}
```

```
export const getters = {}
```

store/auth.jsのstate

ログインしているかどうかの情報をもたせる

```
export const state = () => ({  
  isLoggedIn: false,  
  userId: "",  
  email: ""  
})
```


store/auth.jsのmutations


stateを変更するためのメソッドを用意する

```
export const mutations = {  
  setLoginState( state, loggedIn ){  
    state.isLoggedIn = loggedIn  
  },  
  setUserUid( state, userUid ){  
    state.userUid = userUid  
  },  
  setEmail( state, email ){  
    state.email = email  
  }  
}
```

store/auth.jsのgetters

Stateを参照する時はgettersを使う方が好ましい
(誤ってstateを直接置き換えるのを防ぐため)

```
export const getters = {  
  getLoggedIn: state => !!state.isLoggedIn, // !!で真偽値にキャスト  
  getUserId: state => state.userId,  
  getEmail: state => state.email  
}
```

Vuexでログイン

store/auth.jsのactions

actions->mutations->stateの流れを忘れずに

```
import { getAuth, signInWithEmailAndPassword } from 'firebase/auth'
```

```
//略
```

```
export const actions = {
```

```
// 第1引数・・・context.commitか{ commit } 第2引数・・・payload(オブジェクト)
```

```
  async login({ commit }, payload){
```

```
    const auth = getAuth(this.$firebase)
```

```
    await signInWithEmailAndPassword(auth, payload.email, payload.password)
```

```
    .then(( userCredential )=>{
```

```
      commit('setLoginState', true) // mutationsを指定
```

```
      commit('setUserId', userCredential.user.uid) // mutationsを指定
```

```
      commit('setEmail', userCredential.user.email) // mutationsを指定
```

```
      console.log('ログインok!')
```


```
      this.$router.push('/book')) // bookに移動
```

```
    .catch(e => alert(e))
```

```
  }
```


pages/auth/login.vue

```
// 略
<script>
// 略
  methods:{
    login(){
      this.$store.dispatch('auth/login', {
        email: this.email,
        password: this.password
      })
    }
  }
</script>
```

Vuexでログアウト

store/auth.jsのactions

```
// signOutを追記
import { signOut } from 'firebase/auth'
//略
async logout({ commit }){
  const auth = getAuth(this.$firebase)
  await signOut(auth).then(()=>{
    commit('setLoginState', false)
    commit('setUserId', '')
    commit('setEmail', '')
    this.$router.push('/auth/login')
  }).catch(e => alert(e) )
}
```

クリックでログアウト

components/Header.vue

// 略

items: [

{

title: 'logout',

to: '/auth/logout' // 一旦ページに飛ばす

}

]

pages/auth/logout.vue

// 略

created(){

 this.\$store.dispatch('auth/logout')

}

ログインページのメニュー非表示

Components/Header.vue

// 略

```
<div v-show="isLoggedIn">  
  <v-app-bar-nav-icon @click.stop="drawer = !drawer" />  
</div>
```

// 略

```
computed:{  
  isLoggedIn: () => this.$store.getters['auth/getLoggedIn']  
  // gettersは[]で指定必要なので注意  
}
```




middlewareを活用

未ログインならリダイレクト

middleware/authenticated.js を作成

nuxt.config.jsのrouterに追加

// 略

```
router: {  
  middleware: 'authenticated'  
}
```

これで全ページで実行される

middleware/authenticated.js

```
// middlewareはNuxtの機能なので contextが使える
// pluginは$をつけて取得できる
export default function({
  $firebase, store, route, redirect
})
{
  const isAuthenticated = store.getters['auth/getLoggedIn']
  // string.match(/文字列/) の文字列を含むかのチェック
  if( !isAuthenticated && !route.path.match(/\//auth\//) ){
    redirect('/auth/login')
  }
}
```


リロードでstoreが消える問題



画面表示前に
onAuthStateChanged()を実行
ログイン状態であればstateに保存

middleware/authenticated.js

```
import { getAuth, onAuthStateChanged } from 'firebase/auth'

export default function ({
  略
}) {
  const auth = getAuth($firebase)
  if(!store.getters['auth/getLoggedIn']){
    onAuthStateChanged(auth, user => {
      if (user) {
        store.dispatch('auth/addUserInfo', user)
      } else if(!route.path.match(/\//auth\//)) {
        redirect('/auth/login') }    })}
}
```


store/auth.jsのactions

画面読込時にstateに登録するようのactionを作成

```
export const actions = {
```

略

```
  addUserInfo({ commit }, payload) {  
    commit('setLoginState', true)  
    commit('setUserUid', payload.uid)  
    commit('setEmail', payload.email)  
  }
```

```
}
```




Hosting

hostingの設定



Firestore CLIのインストール

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

```
$firebase login // Googleアカウントで認証
```

```
$firebase init // プロジェクト初期設定
```

firebase init

Hosting: Configure files for Firebase Hosting
and～ をスペースで選択しEnter
Use an existing projectを選択
Firebaseのプロジェクトを選択
生成するフォルダ-> publicではなく dist
Single-pageか・・・Yes
GitHubへの自動ビルド設定するか・・・No
dist/index.htmlを上書きするか・・・Yes

firebase.json と .firebaserc ファイルが生成される

起動確認・デプロイ

ローカルで確認

firebase serve —only hosting

公開用ファイルを生成しておく必要あり

npm run generate

- ・ distフォルダ内に生成

デプロイ

firebase deploy